COM and .Net (Windows)

Introduction

COM is an acronym for Component Object Model; it is an object orientated layer (and associated services) on top of DCE RPC (an open standard) and defines a common calling convention that enables code written in any language to call and interoperate with code written in any other language (provided those languages are COM aware). Not only can the code be written in any language, but it need not even be part of the same executable; the code can be loaded from a DLL, be found in another process running on the same machine, or, with DCOM (Distributed COM), be found in another process on a remote machine, all without your code even needing to know where a component resides.

There is a subset of COM known as OLE Automation which comprises a set of COM interfaces that allow loose binding to COM objects, so that they can be introspected and called at run-time without compile-time knowledge of how the object works. The PHP COM extension utilizes the OLE Automation interfaces to allow you to create and call compatible objects from your scripts. Technically speaking, this should really be called the "OLE Automation Extension for PHP", since not all COM objects are OLE compatible.

Now, why would or should you use COM? COM is one of the main ways to glue applications and components together on the Windows platform; using COM you can launch Microsoft Word, fill in a document template and save the result as a Word document and send it to a visitor of your web site. You can also use COM to perform administrative tasks for your network and to configure your IIS; these are just the most common uses; you can do much more with COM.

Starting with PHP 5, this extension (and this documentation) was rewritten from scratch and much of the old confusing and bogus cruft has be removed. Additionally, we support the instantiation and creation of .Net assemblies using the COM interoperability layer provided by Microsoft.

Please read » this article for an overview of the changes in this extension in PHP 5.

Installing/Configuring

COM functions are only available for the Windows version of PHP.

.Net support requires PHP 5 and the .Net runtime.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

The Windows version of PHP has built-in support for this extension. You do not need to load any additional extensions in order to use these functions.

You are responsible for installing support for the various COM objects that you intend to use (such as MS Word); we don't and can't bundle all of those with PHP.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

Com configuration options

Name	Default	Changeable	Changelog
com.allow_dcom	"0"	PHP_INI_SYSTEM	Available since PHP 4.0.5.
com.autoregister_typelib	"0"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP 4. Available since PHP 4.1.0.
com.autoregister_verbose	"0"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP 4. Available since PHP 4.1.0.
com.autoregister_casesensitive	"1"	PHP_INI_ALL	PHP_INI_SYSTEM in PHP 4. Available since PHP 4.1.0.
com.code_page	""	PHP_INI_ALL	Available since PHP 5.0.0.
com.typelib_file	""	PHP_INI_SYSTEM	Available since PHP 4.0.5.

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

com.allow_dcom
When this is turned on, PHP will be allowed to operate as a D-COM (Distributed COM) client and will allow the PHP script to instantiate COM objects on a remote server.

com.autoregister_typelib
When this is turned on, PHP will attempt to register constants from the typelibrary of objects that it instantiates, if those objects implement the interfaces required to obtain that information. The case sensitivity of the constants it registers is controlled by the configuration directive.

com.autoregister_verbose
When this is turned on, any problems with loading a typelibrary during object instantiation will be reported using the PHP error mechanism. The default is off, which does not emit any indication if there was an error finding or loading the type library.

com.autoregister_casesensitive
When this is turned on (the default), constants found in auto-loaded type libraries will be registered case sensitively. See com_load_typelib() for more details.

com.code_page
It controls the default character set code-page to use when passing strings to and from COM objects. If set to an empty string, PHP will assume that you want **CP_ACP**, which is the default system ANSI code page. If the text in your scripts is encoded using a different encoding/character set by default, setting this directive will save you from having to pass the code page as a parameter to the COM class constructor. Please note that by using this directive (as with any PHP configuration directive), your PHP script becomes less portable; you should use the COM constructor parameter whenever possible.

Note
This configuration directive was introduced with PHP 5.

com.typelib_file
When set, this should hold the path to a file that contains a list of typelibraries that should be loaded on startup. Each line of the file will be treated as the type library name and loaded as though you had called com_load_typelib(). The constants will be registered persistently, so that the library only needs to be loaded once. If a type library name ends with the string *#cis* or *#case_insensitive*, then the constants from that library will be registered case insensitively.

Resource Types

This extension defines a reference to a COM component returned by deprecated com_load() (this function does not exist in PHP 5; use the COM class instead).

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

CLSCTX_INPROC_SERVER (integer)

CLSCTX_INPROC_HANDLER (integer)

CLSCTX_LOCAL_SERVER (integer)

CLSCTX_REMOTE_SERVER (integer)

CLSCTX_SERVER (integer)

CLSCTX_ALL (integer)

VT_NULL (integer)

VT_EMPTY (integer)

VT_UI1 (integer)

VT_I2 (integer)

VT_I4 (integer)

VT_R4 (integer)

VT_R8 (integer)

VT_BOOL (integer)

VT_ERROR (integer)

VT_CY (integer)

VT_DATE (integer)

VT_BSTR (integer)

VT_DECIMAL (integer)

VT_UNKNOWN (integer)

VT_DISPATCH (integer)

VT_VARIANT (integer)

VT_I1 (integer)

VT_UI2 (integer)

VT_UI4 (integer)

VT_INT (integer)

VT_UINT (integer)

VT_ARRAY (integer)

VT_BYREF (integer)

CP_ACP (integer)

CP_MACCP (integer)

CP_OEMCP (integer)

CP_UTF7 (integer)

CP_UTF8 (integer)

CP_SYMBOL (integer)

CP_THREAD_ACP (integer)

VARCMP_LT (integer)

VARCMP_EQ (integer)

VARCMP_GT (integer)

VARCMP_NULL (integer)

NORM_IGNORECASE (integer)

NORM_IGNORENONSPACE (integer)

NORM_IGNORESYMBOLS (integer)

NORM_IGNOREWIDTH (integer)

NORM_IGNOREKANATYPE (integer)

NORM_IGNOREKASHIDA (integer)

DISP_E_DIVBYZERO (integer)

DISP_E_OVERFLOW (integer)

MK_E_UNAVAILABLE (integer)

Errors and error handling

Exceptions (PHP 5)

This extension will throw instances of the class *com_exception* whenever there is a potentially fatal error reported by COM. All COM exceptions have a well-defined *code* property that corresponds to the HRESULT return value from the various COM operations. You may use this code to make programmatic decisions on how to handle the exception.

Examples

For Each

Starting with PHP 5, you may use PHP's own foreach statement to iterate over the contents of a standard COM/OLE IEnumVariant. In laymans terms, this means that you can use foreach in places where you would have used *For Each* in VB/ASP code.

Example #1 - For Each in ASP

```
<%
Set domainObject = GetObject("WinNT://Domain")
For Each obj in domainObject
 Response.Write obj.Name & "<br />"
Next
%>
```

Example #2 - while() ... Next() in PHP 4

```
<?php
$domainObject = new COM("WinNT://Domain");
while ($obj = $domainObject->Next()) {
  echo $obj->Name . "<br />";
}
?>
```

Example #3 - foreach in PHP 5

```
<?php
$domainObject = new COM("WinNT://Domain");
foreach ($domainObject as $obj) {
  echo $obj->Name . "<br />";
}
?>
```

Arrays and Array-style COM properties

Many COM objects expose their properties as arrays, or using array-style access. In PHP

4, you may use PHP array syntax to read/write such a property, but only a single dimension is allowed. If you want to read a multi-dimensional property, you could instead make the property access into a function call, with each parameter representing each dimension of the array access, but there is no way to write to such a property.

PHP 5 introduces the following new features to make your life easier:

- Access multi-dimensional arrays, or COM properties that require multiple parameters using PHP array syntax. You can also write or set properties using this technique.
- Iterate SafeArrays ("true" arrays) using the foreach control structure. This works because SafeArrays include information about their size. If an array-style property implements IEnumVariant then you can also use foreach for that property too; take a look at For Each for more information on this topic.

COM Functions

See Also

For further information on COM read the » COM specification or perhaps take a look at Don Box's » Yet Another COM Library (YACL). You might find some additional useful information in our FAQ for PHP and COM. If you're thinking of using MS Office applications on the server side, you should read the information here: » Considerations for Server-Side Automation of Office.

COM

COM -- COM class

```
$obj = new COM("Application.ID")
```

Description

The COM class allows you to instantiate an OLE compatible COM object and call its methods and access its properties.

Methods

com **COM::COM** (string `$module_name` [, mixed `$server_name` [, int `$codepage` [, string `$typelib`]]])

COM class constructor. The parameters have the following meanings:

module_name

Can be a ProgID, Class ID or Moniker that names the component to load. A ProgID is typically the application or DLL name, followed by a period, followed by the object name. e.g: *Word.Application*. A Class ID is the UUID that uniquely identifies a given class. A Moniker is a special form of naming, similar in concept to a URL scheme, that identifies a resource and specifies how it should be loaded. As an example, you could load up Word and get an object representing a word document by specifying the full path to the word document as the module name, or you can use *LDAP:* as a moniker to use the ADSI interface to LDAP.

server_name

The name of the DCOM server on which the component should be loaded and run. If **NULL**, the object is run using the default for the application. The default is typically to run it on the local machine, although the administrator might have configured the application to launch on a different machine. If you specify a non- **NULL** value for server, PHP will refuse to load the object unless the configuration option is set to **TRUE**. If *`server_name`* is an array, it should contain the following elements (case sensitive!). Note that they are all optional (although you need to specify both Username and Password together); if you omit the Server setting, the default server will be used (as mentioned above), and the instantiation of the object will not be affected by the directive.

DCOM server name

`server_name` **key**	**type**	**description**
Server	string	The name of the server.
Username	string	The username to connect as.

Password	string	The password for *Username*.
Flags	integer	One or more of the following constants, logically OR'd together: **CLSCTX_INPROC_SERVER**, **CLSCTX_INPROC_HANDLER**, **CLSCTX_LOCAL_SERVER**, **CLSCTX_REMOTE_SERVER**, **CLSCTX_SERVER** and **CLSCTX_ALL**. The default value if not specified here is **CLSCTX_SERVER** if you also omit *Server*, or **CLSCTX_REMOTE_SERVER** if you do specify a server. You should consult the Microsoft documentation for CoCreateInstance for more information on the meaning of these constants; you will typically never have to use them.

codepage

Specifies the codepage that is used to convert strings to unicode-strings and vice versa. The conversion is applied whenever a PHP string is passed as a parameter or returned from a method of this COM object. The code page is sticky in PHP 5, which means that it will propagate to objects and variants returned from the object. Possible values are **CP_ACP** (use system default ANSI code page - the default if this parameter is omitted), **CP_MACCP**, **CP_OEMCP**, **CP_SYMBOL**, **CP_THREAD_ACP** (use codepage/locale set for the current executing thread), **CP_UTF7** and **CP_UTF8**. You may also use the number for a given codepage; consult the Microsoft documentation for more details on codepages and their numeric values.

Overloaded Methods

The returned object is an overloaded object, which means that PHP does not see any fixed methods as it does with regular classes; instead, any property or method accesses are passed through to COM.

Starting with PHP 5, PHP will automatically detect methods that accept parameters by reference, and will automatically convert regular PHP variables to a form that can be passed by reference. This means that you can call the method very naturally; you needn't go to any extra effort in your code.

In PHP 4, to pass parameters by reference you need to create an instance of the VARIANT class to wrap the byref parameters.

Pseudo Methods

In PHP versions prior to 5, a number of not very pleasant hacks meant that the following method names were not passed through to COM and were handled directly by PHP. PHP 5 eliminates these things; read the details below to determine how to fix your scripts. These magic method names are case insensitive.

void **COM::AddRef** (void)

Artificially adds a reference count to the COM object.

Warning
You should never need to use this method. It exists as a logical complement to the Release() method below.

void **COM::Release** (void)

Artificially removes a reference count from the COM object.

Warning
You should never need to use this method. Its existence in PHP is a bug designed to work around a bug that keeps COM objects running longer than they should.

Pseudo Methods for Iterating

These pseudo methods are only available if com_isenum() returns **TRUE**, in which case, they hide any methods with the same names that might otherwise be provided by the COM object. These methods have all been eliminated in PHP 5, and you should use For Each instead.

variant **COM::All** (void)

Returns a variant representing a SafeArray that has 10 elements; each element will be an empty/null variant. This function was supposed to return an array containing all the elements from the iterator, but was never completed. Do not use.

variant **COM::Next** (void)

Returns a variant representing the next element available from the iterator, or **FALSE** when there are no more elements.

variant **COM::Prev** (void)

Returns a variant representing the previous element available from the iterator, or **FALSE** when there are no more elements.

void **COM::Reset** (void)

Rewinds the iterator back to the start.

COM examples

Example #4 - COM example (1)

```
<?php
// starting word
$word = new COM("word.application") or die("Unable to instantiate Word");
echo "Loaded Word, version {$word->Version}\n";

//bring it to front
$word->Visible = 1;

//open an empty document
$word->Documents->Add();

//do some weird stuff
$word->Selection->TypeText("This is a test...");
$word->Documents[1]->SaveAs("Useless test.doc");

//closing word
$word->Quit();

//free the object
$word = null;
?>
```

Example #5 - COM example (2)

```
<?php

$conn = new COM("ADODB.Connection") or die("Cannot start ADO");
$conn->Open("Provider=SQLOLEDB; Data Source=localhost;
Initial Catalog=database; User ID=user; Password=password");

$rs = $conn->Execute("SELECT * FROM sometable");    // Recordset

$num_columns = $rs->Fields->Count();
echo $num_columns . "\n";

for ($i=0; $i < $num_columns; $i++) {
   $fld[$i] = $rs->Fields($i);
}

$rowcount = 0;
```

```
while (!$rs->EOF) {
   for ($i=0; $i < $num_columns; $i++) {
       echo $fld[$i]->value . "\t";
   }
   echo "\n";
   $rowcount++;            // increments rowcount
   $rs->MoveNext();
}

$rs->Close();
$conn->Close();

$rs = null;
$conn = null;

?>
```

DOTNET

DOTNET -- DOTNET class

```
$obj = new DOTNET("assembly", "classname")
```

Description

The DOTNET class allows you to instantiate a class from a .Net assembly and call its methods and access its properties.

Methods

string **DOTNET::DOTNET** (string `$assembly_name`, string `$class_name` [, int `$codepage`])

DOTNET class constructor. *`assembly_name`* specifies which assembly should be loaded, and *`class_name`* specifices which class in that assembly to instantiate. You may optionally specify a *`codepage`* to use for unicode string transformations; see the COM class for more details on code pages.

The returned object is an overloaded object, which means that PHP does not see any fixed methods as it does with regular classes; instead, any property or method accesses are passed through to COM and from there to DOTNET. In other words, the .Net object is mapped through the COM interoperability layer provided by the .Net runtime.

Once you have created a DOTNET object, PHP treats it identically to any other COM object; all the same rules apply.

Example #6 - DOTNET example

```
<?php
$stack = new DOTNET("mscorlib", "System.Collections.Stack");
$stack->Push(".Net");
$stack->Push("Hello ");
echo $stack->Pop() . $stack->Pop();
?>
```

Note

You need to install the .Net runtime on your web server to take advantage of this feature.

VARIANT

VARIANT -- VARIANT class

```
$vVar = new VARIANT($var)
```

Description

The VARIANT is COM's equivalent of the PHP zval; it is a structure that can contain a value with a range of different possible types. The VARIANT class provided by the COM extension allows you to have more control over the way that PHP passes values to and from COM.

Methods

object **VARIANT::VARIANT** ([mixed `$value` [, int `$type` [, int `$codepage`]]])

VARIANT class constructor. Parameters:

value
: initial value. if omitted, or set to **NULL** an VT_EMPTY object is created.

type
: specifies the content type of the VARIANT object. Possible values are one of the **VT_XXX** Predefined Constants. In PHP versions prior to PHP 5, you could force PHP to pass a variant object by reference by OR'ing **VT_BYREF** with the *type*. In PHP 5, this hack is not supported; instead, PHP 5 can detect parameters passed by reference automatically; they do not even need to be passed as VARIANT objects. Consult the MSDN library for additional information on the VARIANT type.

codepage
: specifies the codepage that is used to convert strings to unicode. See the parameter of the same name in the COM class for more information.

PHP versions prior to PHP 5 define a number of (undocumented) virtual properties for instances of the VARIANT class; these properties have all been removed in PHP 5 in favour of its more natural syntax; these differences are best highlighted by example:

Example #7 - Variant example, PHP 4.x style

```
<?php
$v = new VARIANT(42);
print "The type is " . $v->type . "<br/>";
print "The value is " . $v->value . "<br/>";
?>
```

Example #8 - Variant example, PHP 5 style

```
<?php
$v = new VARIANT(42);
print "The type is " . variant_get_type($v) . "<br/>";
print "The value is " . $v . "<br/>";
?>
```

The reason for the change is that, internally, the COM extension sees VARIANT, COM and DOTNET classes as the same thing, and the design philosophy for these classes is that all property and member accesses are passed through to COM with no interference. The new syntax is more natural and less effort, and most of the removed virtual properties didn't make any sense in a PHP context in any case.

Note

PHP 5 takes a much simpler approach to handling VARIANTs; when returning a value or fetching a variant property, the variant is converted to a PHP value only when there is a direct mapping between the types that would not result in a loss of information. In all other cases, the result is returned as an instance of the VARIANT class. You can force PHP to convert or evaluate the variant as a PHP native type by using a casting operator explicitly, or implicitly casting to a string by print() ing it. You may use the wide range of variant functions to perform arithmetic operations on variants without forcing a conversion or risking a loss of data.

See also variant_get_type().

com_addref

com_addref -- Increases the components reference counter [deprecated]

Description

void **com_addref** (void)

Increases the components reference counter.

Return Values

No value is returned.

Notes

Warning
You should never need to use this function.

com_create_guid

com_create_guid -- Generate a globally unique identifier (GUID)

Description

string **com_create_guid** (void)

Generates a Globally Unique Identifier (GUID).

A GUID is generated in the same way as DCE UUID's, except that the Microsoft convention is to enclose a GUID in curly braces.

Return Values

Returns the GUID as a string.

See Also

- **uuid_create()** in the PECL uuid extension

com_event_sink

com_event_sink -- Connect events from a COM object to a PHP object

Description

bool **com_event_sink** (variant `$comobject`, object `$sinkobject` [, mixed `$sinkinterface`])

Instructs COM to sink events generated by *`comobject`* into the PHP object *`sinkobject`*.

Be careful how you use this feature; if you are doing something similar to the example below, then it doesn't really make sense to run it in a web server context.

Parameters

`comobject`

`sinkobject`

`sinkobject` should be an instance of a class with methods named after those of the desired dispinterface; you may use com_print_typeinfo() to help generate a template class for this purpose.

`sinkinterface`

PHP will attempt to use the default dispinterface type specified by the typelibrary associated with *`comobject`*, but you may override this choice by setting *`sinkinterface`* to the name of the dispinterface that you want to use.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #9 - COM event sink example

```
<?php
class IEEventSinker {
   var $terminated = false;

  function ProgressChange($progress, $progressmax) {
     echo "Download progress: $progress / $progressmax\n";
   }

   function DocumentComplete(&$dom, $url) {
     echo "Document $url complete\n";
```

```
    }

    function OnQuit() {
      echo "Quit!\n";
      $this->terminated = true;
    }
}
$ie = new COM("InternetExplorer.Application");
// note that you don't need the & for PHP 5!
$sink =& new IEEventSinker();
com_event_sink($ie, $sink, "DWebBrowserEvents2");
$ie->Visible = true;
$ie->Navigate("http://www.php.net");
while(!$sink->terminated) {
 com_message_pump(4000);
}
$ie = null;
?>
```

See Also

- com_print_typeinfo()
- com_message_pump()

com_get_active_object

com_get_active_object -- Returns a handle to an already running instance of a COM object

Description

variant **com_get_active_object** (string `$progid` [, int `$code_page`])

com_get_active_object() is similar to creating a new instance of a COM object, except that it will only return an object to your script if the object is already running. OLE applications use something known as the Running Object Table to allow well-known applications to be launched only once; this function exposes the COM library function GetActiveObject() to get a handle on a running instance.

Parameters

`progid`
: `progid` must be either the ProgID or CLSID for the object that you want to access (for example *Word.Application*).

`code_page`
: Acts in precisely the same way that it does for the COM class.

Return Values

If the requested object is running, it will be returned to your script just like any other COM object.

Errors/Exceptions

There are a variety of reasons why this function might fail, the most common being that the object is not already running. In that situation, the exception error code will be **MK_E_UNAVAILABLE**; you can use the *getCode* method of the exception object to check the exception code.

Notes

Warning
Using com_get_active_object() in a web server context is not always a smart idea. Most COM/OLE applications are not designed to handle more than one client concurrently, even (or especially!) Microsoft Office. You should read » Considerations for Server-Side Automation of Office for more information on the general issues involved.

com_get

com_get -- Gets the value of a COM Component's property [deprecated]

Description

Deprecated, use the OO syntax instead.

Example #10 - OO syntax

```
<?php
// do this
$var = $obj->property;
// instead of this:
$var = com_get($obj, 'property');
?>
```

Notes

Note

This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

com_invoke

com_invoke -- Calls a COM component's method [deprecated]

Description

mixed **com_invoke** (resource `$com_object`, string `$function_name` [, mixed `$function_parameters`])

com_invoke() invokes the method named *`function_name`* of the COM component referenced by *`com_object`*. com_invoke() returns **FALSE** on error, returns the *`function_name`* 's return value on success. All the extra parameters *`function_parameters`* are passed to the method *`function_name`*.

Example #11 - Don't use com_invoke(), use OO syntax instead

```
<?php
// do this
$val = $obj->method($one, $two);
// instead of this:
$val = com_invoke($obj, 'method', $one, $two);
?>
```

Note

This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

com_isenum

com_isenum -- Indicates if a COM object has an IEnumVariant interface for iteration [deprecated]

Description

bool **com_isenum** (variant `$com_module`)

Checks to see if a COM object can be enumerated using the *Next()* method hack. See COM class for more details on these methods.

Parameters

`com_module`
The COM object.

Return Values

Returns **TRUE** if the object can be enumatated, **FALSE** otherwise.

Notes

Note
This function does not exist in PHP 5; use the more natural foreach statement to iterate over the contents of COM objects. See For Each for more details.

com_load_typelib

com_load_typelib -- Loads a Typelib

Description

bool **com_load_typelib** (string `$typelib_name` [, bool `$case_insensitive`])

Loads a type-library and registers its constants in the engine, as though they were defined using define().

Note that it is much more efficient to use the configuration setting to pre-load and register the constants, although not so flexible.

If you have turned on , then PHP will attempt to automatically register the constants associated with a COM object when you instantiate it. This depends on the interfaces provided by the COM object itself, and may not always be possible.

Parameters

`typelib_name`
`typelib_name` can be one of the following:

- The filename of a *.tlb* file or the executable module that contains the type library.
- The type library GUID, followed by its version number, for example *{00000200-0000-0010-8000-00AA006D2EA4},2,0*.
- The type library name, e.g. *Microsoft OLE DB ActiveX Data Objects 1.0 Library*.

PHP will attempt to resolve the type library in this order, as the process gets more and more expensive as you progress down the list; searching for the type library by name is handled by physically enumerating the registry until we find a match.

`case_insensitive`
The `case_insensitive` behaves in the same way as the parameter with the same name in the define() function.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

com_load

com_load -- Creates a new reference to a COM component [deprecated]

Description

Deprecated, use the OO syntax instead.

Example #12 - OO syntax

```
<?php
// do this
$obj = new COM($module);
// instead of this:
$obj = com_load($module);
?>
```

Notes

Note

This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

com_message_pump

com_message_pump -- Process COM messages, sleeping for up to timeoutms milliseconds

Description

bool **com_message_pump** ([int `$timeoutms`])

This function will sleep for up to *`timeoutms`* milliseconds, or until a message arrives in the queue.

The purpose of this function is to route COM calls between apartments and handle various synchronization issues. This allows your script to wait efficiently for events to be triggered, while still handling other events or running other code in the background. You should use it in a loop, as demonstrated by the example in the com_event_sink() function, until you are finished using event bound COM objects.

Parameters

`timeoutms`
: The timeout, in milliseconds. If you do not specify a value for *`timeoutms`*, then 0 will be assumed. A 0 value means that no waiting will be performed; if there are messages pending they will be dispatched as before; if there are no messages pending, the function will return **FALSE** immediately without sleeping.

Return Values

If a message or messages arrives before the timeout, they will be dispatched, and the function will return **TRUE**. If the timeout occurs and no messages were processed, the return value will be **FALSE**.

com_print_typeinfo

com_print_typeinfo -- Print out a PHP class definition for a dispatchable interface

Description

bool **com_print_typeinfo** (object `$comobject` [, string `$dispinterface` [, bool `$wantsink`]])

The purpose of this function is to help generate a skeleton class for use as an event sink. You may also use it to generate a dump of any COM object, provided that it supports enough of the introspection interfaces, and that you know the name of the interface you want to display.

Parameters

`comobject`
: *`comobject`* should be either an instance of a COM object, or be the name of a typelibrary (which will be resolved according to the rules set out in com_load_typelib()).

`dispinterface`
: The name of an IDispatch descendant interface that you want to display.

`wantsink`
: If set to **TRUE**, the corresponding sink interface will be displayed instead.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- com_event_sink()
- com_load_typelib()

com_propget

com_propget -- Alias of com_get()

Description

This function is an alias of: com_get().

Note
This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

com_propput

com_propput -- Alias of com_set()

Description

This function is an alias of: com_set().

Note
This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

com_propset

com_propset -- Alias of com_set()

Description

This function is an alias of: com_set().

Note
This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

com_release

com_release -- Decreases the components reference counter [deprecated]

Description

void **com_release** (void)

Decreases the components reference counter.

Return Values

No value is returned.

ChangeLog

Version	Description
5.0.0	This function was removed.

Notes

Warning
You should never need to use this function.

com_set

com_set -- Assigns a value to a COM component's property

Description

Deprecated, use the OO syntax instead.

Example #13 - OO syntax

```
<?php
// do this
$obj->property = $value;
// instead of this:
com_set($obj, 'property', $value);
?>
```

Notes

Note

This function does not exist in PHP 5; instead, you should use the regular and more natural OO syntax to access properties or call methods.

variant_abs

variant_abs -- Returns the absolute value of a variant

Description

mixed **variant_abs** (mixed `$val`)

Returns the absolute value of a variant.

Parameters

`val`
The variant.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the absolute value of *`val`*.

See Also

- abs()

variant_add

variant_add -- "Adds" two variant values together and returns the result

Description

mixed **variant_add** (mixed `$left`, mixed `$right`)

Adds *`left`* to *`right`* using the following rules (taken from the MSDN library), which correspond to those of Visual Basic:

Variant Addition Rules

If	Then
Both expressions are of the string type	Concatenation
One expression is a string type and the other a character	Addition
One expression is numeric and the other is a string	Addition
Both expressions are numeric	Addition
Either expression is NULL	NULL is returned
Both expressions are empty	Integer subtype is returned

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value.

The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the result.

See Also

- variant_sub()

variant_and

variant_and -- Performs a bitwise AND operation between two variants

Description

mixed **variant_and** (mixed `$left`, mixed `$right`)

Performs a bitwise AND operation. Note that this is slightly different from a regular AND operation.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant AND Rules

If *`left`* **is**	**If** *`right`* **is**	**then the result is**
TRUE	**TRUE**	**TRUE**
TRUE	**FALSE**	**FALSE**

TRUE	**NULL**	**NULL**
FALSE	**TRUE**	**FALSE**
FALSE	**FALSE**	**FALSE**
FALSE	**NULL**	**FALSE**
NULL	**TRUE**	**NULL**
NULL	**FALSE**	**FALSE**
NULL	**NULL**	**NULL**

See Also

- variant_or()

variant_cast

variant_cast -- Convert a variant into a new variant object of another type

Description

variant **variant_cast** (variant `$variant`, int `$type`)

This function makes a copy of *`variant`* and then performs a variant cast operation to force the copy to have the type given by *`type`*.

This function wraps VariantChangeType() in the COM library; consult MSDN for more information.

Parameters

`variant`
The variant.

`type`
`type` should be one of the **VT_XXX** constants.

Return Values

Returns a **VT_DATE** variant.

See Also

- variant_set_type()

variant_cat

variant_cat -- concatenates two variant values together and returns the result

Description

mixed **variant_cat** (mixed `$left`, mixed `$right`)

Concatenates *`left`* with *`right`* and returns the result.

This function is notionally equivalent to *`$left. $right`*.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the result of the concatenation.

See Also

- String Operators for the string concatenation operator

variant_cmp

variant_cmp -- Compares two variants

Description

int **variant_cmp** (mixed `$left`, mixed `$right` [, int `$lcid` [, int `$flags`]])

Compares *`left`* with *`right`*.

This function will only compare scalar values, not arrays or variant records.

Parameters

`left`
The left operand.

`right`
The right operand.

`lcid`
A valid Locale Identifier to use when comparing strings (this affects string collation).

`flags`
`flags` can be one or more of the following values OR'd together, and affects string comparisons:

Variant Comparision Flags

value	meaning
NORM_IGNORECASE	Compare case insensitively
NORM_IGNORENONSPACE	Ignore nonspacing characters
NORM_IGNORESYMBOLS	Ignore symbols
NORM_IGNOREWIDTH	Ignore string width
NORM_IGNOREKANATYPE	Ignore Kana type
NORM_IGNOREKASHIDA	Ignore Arabic kashida characters

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns one of the following:

Variant Comparision Results

value	meaning
VARCMP_LT	*left* is less than *right*
VARCMP_EQ	*left* is equal to *right*
VARCMP_GT	*left* is greater than *right*
VARCMP_NULL	Either *left*, *right* or both are **NULL**

variant_date_from_timestamp

variant_date_from_timestamp -- Returns a variant date representation of a Unix timestamp

Description

variant **variant_date_from_timestamp** (int `$timestamp`)

Converts *`timestamp`* from a unix timestamp value into a variant of type **VT_DATE**. This allows easier interopability between the unix-ish parts of PHP and COM.

Parameters

`timestamp`
: A unix timestamp.

Return Values

Returns a **VT_DATE** variant.

See Also

- variant_date_to_timestamp()
- mktime()
- time()

variant_date_to_timestamp

variant_date_to_timestamp -- Converts a variant date/time value to Unix timestamp

Description

int **variant_date_to_timestamp** (variant `$variant`)

Converts *`variant`* from a **VT_DATE** (or similar) value into a Unix timestamp. This allows easier interopability between the Unix-ish parts of PHP and COM.

Parameters

`variant`
The variant.

Return Values

Returns a unix timestamp.

See Also

- variant_date_from_timestamp()
- date()
- strftime()

variant_div

variant_div -- Returns the result from dividing two variants

Description

mixed **variant_div** (mixed `$left`, mixed `$right`)

Divides *`left`* by *`right`* and returns the result.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant Division Rules

If	**Then**
Both expressions are of the string, date, character, boolean type	Double is returned
One expression is a string type and the other a character	Division and a double is returned

One expression is numeric and the other is a string	Division and a double is returned.
Both expressions are numeric	Division and a double is returned
Either expression is NULL	NULL is returned
right is empty and *left* is anything but empty	A com_exception with code **DISP_E_DIVBYZERO** is thrown
left is empty and *right* is anything but empty.	0 as type double is returned
Both expressions are empty	A com_exception with code **DISP_E_OVERFLOW** is thrown

See Also

- variant_idiv()

variant_eqv

variant_eqv -- Performs a bitwise equivalence on two variants

Description

mixed **variant_eqv** (mixed `$left`, mixed `$right`)

Performs a bitwise equivalence on two variants.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

If each bit in *`left`* is equal to the corresponding bit in *`right`* then **TRUE** is returned, otherwise **FALSE** is returned.

variant_fix

variant_fix -- Returns the integer portion of a variant

Description

mixed **variant_fix** (mixed `$variant`)

Gets the integer portion of a variant.

Parameters

`variant`
The variant.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

If *`variant`* is negative, then the first negative integer greater than or equal to the variant is returned, otherwise returns the integer portion of the value of *`variant`*.

Notes

Warning
This documentation is based on the MSDN documentation; it appears that this function is either the same as variant_int(), or that there is an error in the MSDN documentation.

See Also

- variant_int()
- variant_round()
- floor()
- ceil()
- round()

variant_get_type

variant_get_type -- Returns the type of a variant object

Description

int **variant_get_type** (variant `$variant`)

Returns the type of a variant object.

Parameters

`variant`
The variant object.

Return Values

This function returns an integer value that indicates the type of *`variant`*, which can be an instance of COM, DOTNET or VARIANT classes. The return value can be compared to one of the **VT_XXX** constants.

The return value for COM and DOTNET objects will usually be **VT_DISPATCH**; the only reason this function works for those classes is because COM and DOTNET are descendants of VARIANT.

In PHP versions prior to 5, you could obtain this information from instances of the VARIANT class ONLY, by reading a fake *`type`* property. See the VARIANT class for more information on this.

variant_idiv

variant_idiv -- Converts variants to integers and then returns the result from dividing them

Description

mixed **variant_idiv** (mixed `$left`, mixed `$right`)

Converts *`left`* and *`right`* to integer values, and then performs integer division.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant Integer Division Rules

If	**Then**
Both expressions are of the string, date, character, boolean type	Division and integer is returned
One expression is a string type and the other a character	Division

One expression is numeric and the other is a string	Division
Both expressions are numeric	Division
Either expression is NULL	NULL is returned
Both expressions are empty	A com_exception with code **DISP_E_DIVBYZERO** is thrown

See Also

- variant_div()

variant_imp

variant_imp -- Performs a bitwise implication on two variants

Description

mixed **variant_imp** (mixed `$left`, mixed `$right`)

Performs a bitwise implication operation.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant Implication Table

If *`left`* **is**	**If** *`right`* **is**	**then the result is**
TRUE	**TRUE**	**TRUE**
TRUE	**FALSE**	**TRUE**
TRUE	**NULL**	**TRUE**

FALSE	**TRUE**	**TRUE**
FALSE	**FALSE**	**TRUE**
FALSE	**NULL**	**TRUE**
NULL	**TRUE**	**TRUE**
NULL	**FALSE**	**NULL**
NULL	**NULL**	**NULL**

variant_int

variant_int -- Returns the integer portion of a variant

Description

mixed **variant_int** (mixed `$variant`)

Gets the integer portion of a variant.

Parameters

`variant`
The variant.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

If *`variant`* is negative, then the first negative integer greater than or equal to the variant is returned, otherwise returns the integer portion of the value of *`variant`*.

See Also

- variant_fix()
- variant_round()
- floor()
- ceil()
- round()

variant_mod

variant_mod -- Divides two variants and returns only the remainder

Description

mixed **variant_mod** (mixed `$left`, mixed `$right`)

Divides *`left`* by *`right`* and returns the remainder.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the remainder of the division.

See Also

- variant_div()
- variant_idiv()

variant_mul

variant_mul -- Multiplies the values of the two variants

Description

mixed **variant_mul** (mixed `$left`, mixed `$right`)

Multiplies *`left`* by *`right`*.

Parameters

`left`
The left operand.

`right`
The right operand.

Boolean values are converted to -1 for **FALSE** and 0 for **TRUE**.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant Multiplication Rules

If	**Then**
Both expressions are of the string, date, character, boolean type	Multiplication
One expression is a string type and the other a character	Multiplication

One expression is numeric and the other is a string	Multiplication
Both expressions are numeric	Multiplication
Either expression is NULL	NULL is returned
Both expressions are empty	Empty string is returned

See Also

- variant_div()
- variant_idiv()

variant_neg

variant_neg -- Performs logical negation on a variant

Description

mixed **variant_neg** (mixed `$variant`)

Performs logical negation of *`variant`*.

Parameters

`variant`
The variant.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the result of the logical negation.

variant_not

variant_not -- Performs bitwise not negation on a variant

Description

mixed **variant_not** (mixed `$variant`)

Performs bitwise not negation on *`variant`* and returns the result.

Parameters

`variant`
The variant.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the bitwise not negation. If *`variant`* is **NULL**, the result will also be **NULL**.

variant_or

variant_or -- Performs a logical disjunction on two variants

Description

mixed **variant_or** (mixed `$left`, mixed `$right`)

Performs a bitwise OR operation. Note that this is slightly different from a regular OR operation.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant OR Rules

If *`left`* **is**	**If** *`right`* **is**	**then the result is**
TRUE	**TRUE**	**TRUE**
TRUE	**FALSE**	**TRUE**

TRUE	**NULL**	**TRUE**
FALSE	**TRUE**	**TRUE**
FALSE	**FALSE**	**FALSE**
FALSE	**NULL**	**NULL**
NULL	**TRUE**	**TRUE**
NULL	**FALSE**	**NULL**
NULL	**NULL**	**NULL**

See Also

- variant_and()
- variant_xor()

variant_pow

variant_pow -- Returns the result of performing the power function with two variants

Description

mixed **variant_pow** (mixed `$left`, mixed `$right`)

Returns the result of *`left`* to the power of *`right`*.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the result of *`left`* to the power of *`right`*.

See Also

- pow()

variant_round

variant_round -- Rounds a variant to the specified number of decimal places

Description

mixed **variant_round** (mixed `$variant`, int `$decimals`)

Returns the value of *`variant`* rounded to *`decimals`* decimal places.

Parameters

`variant`
The variant.

`decimals`
Number of decimal places.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Returns the rounded value.

See Also

- round()

variant_set_type

variant_set_type -- Convert a variant into another type "in-place"

Description

void **variant_set_type** (variant `$variant`, int `$type`)

This function is similar to variant_cast() except that the variant is modified "in-place"; no new variant is created. The parameters for this function have identical meaning to those of variant_cast().

Parameters

`variant`
The variant.

`type`

Return Values

No value is returned.

See Also

- variant_cast()

variant_set

variant_set -- Assigns a new value for a variant object

Description

void **variant_set** (variant `$variant`, mixed `$value`)

Converts *`value`* to a variant and assigns it to the *`variant`* object; no new variant object is created, and the old value of *`variant`* is freed/released.

Parameters

`variant`
The variant.

`value`

Return Values

No value is returned.

variant_sub

variant_sub -- Subtracts the value of the right variant from the left variant value

Description

mixed **variant_sub** (mixed `$left`, mixed `$right`)

Subtracts *`right`* from *`left`*.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant Subtraction Rules

If	**Then**
Both expressions are of the string type	Subtraction
One expression is a string type and the other a character	Subtraction

One expression is numeric and the other is a string	Subtraction.
Both expressions are numeric	Subtraction
Either expression is NULL	NULL is returned
Both expressions are empty	Empty string is returned

See Also

- variant_add()

variant_xor

variant_xor -- Performs a logical exclusion on two variants

Description

mixed **variant_xor** (mixed `$left`, mixed `$right`)

Performs a logical exclusion.

Parameters

`left`
The left operand.

`right`
The right operand.

Note
As with all the variant arithmetic functions, the parameters for this function can be either a PHP native type (integer, string, floating point, boolean or **NULL**), or an instance of a COM, VARIANT or DOTNET class. PHP native types will be converted to variants using the same rules as found in the constructor for the VARIANT class. COM and DOTNET objects will have the value of their default property taken and used as the variant value. The variant arithmetic functions are wrappers around the similarly named functions in the COM library; for more information on these functions, consult the MSDN library. The PHP functions are named slightly differently; for example variant_add() in PHP corresponds to *VarAdd()* in the MSDN documentation.

Return Values

Variant XOR Rules

If *`left`* **is**	**If** *`right`* **is**	**then the result is**
TRUE	**TRUE**	**FALSE**
TRUE	**FALSE**	**TRUE**
FALSE	**TRUE**	**TRUE**

FALSE	**FALSE**	**FALSE**
NULL	**NULL**	**NULL**

See Also

- variant_or()
- variant_and()