Service Data Objects

Introduction

Service Data Objects (SDOs) enable PHP applications to work with data from different sources (like a database query, an XML file, and a spreadsheet) using a single interface.

Each different kind of data source requires a Data Access Service (DAS) to provide access to the data in the data source. In your PHP application, you use a DAS to create an SDO instance that represents some data in the data source. You can then set and get values in the SDO instance using the standard SDO interface. Finally, you use a DAS to write the modified data back to a data source, typically the same one.

See the list of Data Access Services for details on those currently available. In addition to the provided DASs, SDO also provides interfaces to enable others to be implemented (see the section on SDO Data Access Services Interface for more details).

This extension is derived from concepts taken from the » Service Data Objects specification. It includes a version of the » Apache Tuscany SDO for C++ project.

The Structure of a Service Data Object

A Service Data Object instance is made up of a tree of data objects. The tree is defined by containment relationships between the data objects. For example, a Company data object might consist of a number of Department data objects and therefore the Company would have a containment relationship to the Departments.

An SDO may also have non-containment references between data objects in the tree. For example, one Employee data object might reference another Employee to identify a career mentor.

As well as data objects referencing each other, they can also have primitive properties. For example, the Company data object might have a property called "name" of type string, for holding the name of the company (for example, "Acme").

Each of these properties of a data object - containment relationships, non-containment references, or primitive properties - may be many-valued or single-valued. In the above examples, Departments is many-valued and the Company name is single-valued.

In PHP, each SDO data object is represented as a PHP object. The properties of the data object can be accessed using either object syntax or associative array syntax. We'll see some examples of this later.

Installing/Configuring

Requirements

The SDO extension requires PHP 5.1.0 or higher. It also requires the libxml2 library. Normally libxml2 will already be installed, but if not, it can be downloaded from » http://www.xmlsoft.org/.

Installation

Note
Earlier versions of the SDO extension required a separate shared library for the XML DAS. This is now obsolete and any references to *php_sdo_das_xml.dll* or *sdo_das_xml.so* should be removed from your *php.ini*.

Unix systems

1. The three SDO components - the SDO core, the XML DAS and the Relational DAS - are packaged together with Service Component Architecture (SCA) into one PECL project, SCA_SDO, so you can download SCA and all three parts of SDO with the command:
   ```
   pecl install SCA_SDO
   ```
   This command will build the SDO shared library as well as installing the PHP files that make up SCA and the SDO Relational DAS. If you want to use the latest beta version, then instead run:
   ```
   pecl install SCA_SDO-beta
   ```

2. The *pecl* command automatically installs the SDO module into your PHP extensions directory. To enable the SDO extension you must add the following line to *php.ini*:
   ```
   extension=sdo.so
   ```
   For more information about building PECL packages, consult the PECL installation section of the manual.

Windows

1. The latest SDO DLL can be downloaded from » php_sdo.dll. Note that currently the » pecl4win site does not provide this binary at the current release level; you can only download the latest level.
2. The *pecl* command automatically installs the SDO module into your PHP extensions directory. To enable the SDO extension you must add the following line to *php.ini*:
   ```
   extension=php_sdo.dll
   ```
3. The Relational DAS can be downloaded and installed with the command:
   ```
   pecl install -B sdo
   ```
   The Relational DAS is written in PHP. You may need to update your include_path in

php.ini to point to the directory that contains *sdo/DAS/Relational*.

Building SDO on Linux
This section describes how to build the SDO core and XML DAS on Linux. You would only need to know how to do this if you wish to build a recent version that you have checked out of CVS.

1. Change to the main extension directory: *cd < wherever your sdo code is >*
2. Run *phpize*, which will set up the environment to compile SDO.
3. Next, run*./configure; make; make install*. Please note, you may need to login as root to install the extension.
4. Make sure that the module is loaded by PHP, by adding *extension=sdo.so* to your *php.ini* file.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

This extension has no resource types defined.

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.

SDO_DAS_ChangeSummary::NONE=0 (integer)
Represents a change type of 'none'.

SDO_DAS_ChangeSummary::MODIFICATION=1 (integer)
Represents a change type of 'modification'.

SDO_DAS_ChangeSummary::ADDITION=2 (integer)
Represents a change type of 'addition'.

SDO_DAS_ChangeSummary::DELETION=3 (integer)
Represents a change type of 'deletion'.

Limitations

Implementation Limitations

The following are limitations in the current SDO implementation:

1. There is no support for multi-byte character sets. This will be considered, depending on community requirements, in the Unicode-enabled version of PHP. See Unicode Functions.

SDO Limitations

The following SDO 2.0 concepts are not supported in the current PHP implementation. It is not necessarily the case that these will all be added over time. Their inclusion will depend on community requirements.

1. Bi-directional relationships.
2. Type and property alias names.
3. Read-only properties.
4. The Helper classes defined in SDO 2.0 are not directly implemented. However equivalent function is provided in a more natural way for PHP. For example the function of *CopyHelper::copy()* is provided by applying the PHP clone keyword to a data object.

Examples

Basic Usage

The examples below assume an SDO created with the schema and instance information shown below, using the XML Data Access Service.

The instance document below describes a single company, called 'MegaCorp', which contains a single department, called 'Advanced Technologies'. The Advanced Technologies department contains three employees. The company employeeOfTheMonth is referencing the second employee, 'Jane Doe'.

```
<?xml version="1.0" encoding="UTF-8" ?>
<company xmlns="companyNS" name="MegaCorp"
 employeeOfTheMonth="E0003">
 <departments name="Advanced Technologies" location="NY" number="123">
   <employees name="John Jones" SN="E0001"/>
   <employees name="Jane Doe" SN="E0003"/>
   <employees name="Al Smith" SN="E0004" manager="true"/>
 </departments>
</company>
```

The root element of the schema is a company. The company contains departments, and each department contains employees. Each element has a number of attributes to store things like name, serial number, and so on. Finally, the company also has an IDREF attribute which identifies one of the employees as the 'employeeOfTheMonth'.

```
<xsd:schema
 xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:sdo="commonj.sdo"
 xmlns:sdoxml="commonj.sdo/xml"
 xmlns:company="companyNS"
 targetNamespace="companyNS">
 <xsd:element name="company" type="company:CompanyType"/>
 <xsd:complexType name="CompanyType">
   <xsd:sequence>
     <xsd:element name="departments" type="company:DepartmentType"
       maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string"/>
   <xsd:attribute name="employeeOfTheMonth" type="xsd:IDREF"
     sdoxml:propertyType="company:EmployeeType"/>
 </xsd:complexType>
 <xsd:complexType name="DepartmentType">
   <xsd:sequence>
     <xsd:element name="employees" type="company:EmployeeType"
       maxOccurs="unbounded"/>
   </xsd:sequence>
   <xsd:attribute name="name" type="xsd:string"/>
   <xsd:attribute name="location" type="xsd:string"/>
```

```
   <xsd:attribute name="number" type="xsd:int"/>
 </xsd:complexType>
 <xsd:complexType name="EmployeeType">
   <xsd:attribute name="name" type="xsd:string"/>
   <xsd:attribute name="SN" type="xsd:ID"/>
   <xsd:attribute name="manager" type="xsd:boolean"/>
 </xsd:complexType>
</xsd:schema>
```

The XML Data Access Service maps the schema to an SDO. Attributes such as "name" become primitive properties, the sequence of employees becomes a many-valued containment relationship, and so on. Note that the containment relationships are expressed as one complex type within another, whereas non-containment references are expressed in terms of ID and IDREF, with a special *sdoxml:propertyType* attribute specifying the type of the non-containment reference.

Setting and Getting Property Values

The following examples assume *$company* is the root of a tree of data objects created from the schema and instance document shown above.

Example #1 - Access via property name

Data object properties can be accessed using the object property access syntax. The following sets the company name to 'Acme'.

```
<?php
$company->name = 'Acme';
?>
```

Example #2 - Access via property name as array index

We can also access properties using associative array syntax. The simplest form of this uses the property name as the array index. For example, the following sets the company name and gets the employeeOfTheMonth.

```
<?php
$company['name'] = 'UltraCorp';
$eotm = $company['employeeOfTheMonth'];
?>
```

Example #3 - Data Object iteration

We can iterate over the properties of a data object using foreach. The following iterates over the properties of the employee of the month.

```
<?php
 $eotm = $company->employeeOfTheMonth;
 foreach ($eotm as $name => $value) {
     echo "$name: $value\n";
 }
?>
```

which will output:

```
name: Jane Doe
SN: E0003
```

The 'manager' property is not output, because it has not been set.

Example #4 - Access many-valued property by name

Many-valued data object properties can also be accessed using the object property name syntax. The following gets the list of departments.

```
<?php
$departments = $company->departments;
?>
```

Example #5 - Many-valued element access

We can access individual elements of many-valued properties using array syntax. The following accesses the first department in the company.

```
<?php
$ad_tech_dept = $company->departments[0];
?>
```

Example #6 - Many-valued property iteration

Many-valued properties can also be iterated over using foreach. The following iterates over the company's departments.

```
<?php
 foreach ($company->departments as $department) {
   // ...
 }
?>
```

Each iteration will assign the next department in the list to the variable *$department*.

Example #7 - Chained property access

We can chain property references on a single line. The following sets and gets the name of the first department.

```
<?php
 $company->departments[0]->name = 'Emerging Technologies';
 $dept_name = $company->departments[0]->name;
?>
```

Using the associative array syntax, this is equivalent to

```
<?php
 $company['departments'][0]['name'] = 'Emerging Technologies';
 $dept_name = $company['departments'][0]['name'];
?>
```

In either case, the dept_name variable is set to 'Emerging Technologies'.

Example #8 - XPath navigation

The associative array index can be an XPath-like expression. Valid expressions are defined by an augmented sub-set of XPath.

Two forms of indexing into many-valued properties are supported. The first is the standard XPath array syntax with the indexing starting at one, the second is an SDO extension to XPath with an index starting at zero. The standard syntax is:

```
<?php
 $jane_doe = $company["departments[1]/employees[2]"];
?>
```

and the SDO XPath extension syntax is:

```
<?php
 $jane_doe = $company["departments.0/employees.1"];
?>
```

Both these examples get the second employee from the first department.

Example #9 - XPath querying

We can use XPath to query and identify parts of a data object based on instance data. The following retrieves the manager from the 'Advanced Technologies' department.

```
<?php
$ad_tech_mgr =
 $company["departments[name='Advanced
Technologies']/employees[manager=true]"];
?>
```

Example #10 - Creating child data objects

A data object can be a factory for its child data objects. A child data object is automatically part of the data graph. The following add a new employee to the 'Advanced Technologies' department.

```
<?php
 $ad_tech_dept = $company["departments[name='Advanced Technologies']"];
 $new_hire = $ad_tech_dept->createDataObject('employees');
 $new_hire->name = 'John Johnson';
 $new_hire->SN = 'E0005';
 $new_hire->manager = false;
?>
```

Example #11 - Unset a primitive property

We can use the isset() and unset() functions to test and remove items from the data object.

The following clears the name of the first department.

```
<?php
 unset($company->departments[0]->name);
?>
```

Example #12 - Unset a data object

unset can also be used to remove a data object from the tree. The following example shows John Jones leaving the company.

```
<?php
 unset($company->departments[0]->employees[0]);
?>
```

Example #13 - Unset a referenced data object

The following removes the 'employeeOfTheMonth' from the company. If this were a containment relationship then the employee would be removed from the company (probably not a good idea to sack your best employee each month!), but since this is a non-containment reference, the employee being referenced will remain in the department in the company, but will no longer be accessible via the employeeOfTheMonth property.

```
<?php
 if (isset($company->employeeOfTheMonth)) {
   unset($company->employeeOfTheMonth);
 }
?>
```

Example #14 - Access via property index

Data object properties can be accessed via their property index using array syntax. The property index is the position at which the property's definition appears in the model (in this case the xml schema). We can see from the schema listing above that the company name attribute is the second company property (the SDO interface makes no distinction between XML attributes and elements). The following sets the company name to 'Acme', with the same result as Access via property name

```
<?php
 $company[1] = 'Acme';
?>
```

Using the index directly in this way is likely to be fragile. Normally the property name syntax should be preferred, but the property index may be required in special cases.

Working with Sequenced Data Objects

Sequenced data objects are SDOs which can track property ordering across the properties of a data object. They can also contain unstructured text elements (text element which do not belong to any of the SDO's properties). Sequenced data objects are useful for working with XML documents which allow unstructured text (i.e. mixed=true) or if the elements can be interleaved (<A/><B/><A/>). This can occur for example when the schema defines

maxOccurs>1 on a element which is a complexType with a choice order indicator.

The examples below assume an SDO created with the following schema and instance information, using the XML Data Access Service.

The schema below describes the format of a letter. The letter can optionally contain three properties; date, firstName, and lastName. The schema states *mixed="true"* which means that unstructured text can be interspersed between the three properties.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
 xmlns:letter="http://letterSchema"
 targetNamespace="http://letterSchema">
 <xsd:element name="letters" type="letter:FormLetter"/>
 <xsd:complexType name="FormLetter" mixed="true">
   <xsd:sequence>
     <xsd:element name="date" minOccurs="0" type="xsd:string"/>
     <xsd:element name="firstName" minOccurs="0" type="xsd:string"/>
     <xsd:element name="lastName" minOccurs="0" type="xsd:string"/>
   </xsd:sequence>
 </xsd:complexType>
</xsd:schema>
```

The following is an instance letter document. It contains the three letter properties; date, firstName and lastName, and has unstructured text elements for the address and letter body.

```
<letter:letters xmlns:letter="http://letterSchema">
 <date>March 1, 2005</date>
 Mutual of Omaha
 Wild Kingdom, USA
 Dear
 <firstName>Casy</firstName>
 <lastName>Crocodile</lastName>
 Please buy more shark repellent.
 Your premium is past due.
</letter:letters>
```

When loaded, the letter data object will have the sequence and property indices shown in the table below:

Sequence Index	**Property Index:Name**	**Value**
0	0:date	March 1, 2005
1	-	Mutual of Omaha
2	-	Wild Kingdom, USA
3	-	Dear
4	1:firstName	Casy

5	2:lastName	Crocodile
6	-	Please buy more shark repellent.
7	-	Your premium is past due.

To ensure sequence indices are maintained, sequenced data objects should be manipulated through the SDO_Sequence interface. This allows the data object's instance data to be manipulated in terms of the sequence index as opposed to the property index (shown in the table above). The following examples assume the letter instance has been loaded into a data object referenced by the variable *$letter*.

Example #15 - Getting the SDO_Sequence interface

We obtain a data object's sequence using the **getSequence()** method. The follow gets the sequence for the letter data object.

```
<?php
 $letter_seq = $letter->getSequence();
?>
```

All subsequent examples assume that the *$letter_seq* variable has been assigned the sequence for the letter data object.

Example #16 - Get/set sequence values

We can get and set individual values (including unstructured text) using the sequence index. The following sets the firstName to 'Snappy' and gets the last sequence values (the unstructured text, 'Your premium is past due.').

```
<?php
 $letter_seq[4] = 'Snappy';
 $text = $letter_seq[count($letter_seq) - 1];
?>
```

Example #17 - Sequence iteration

We can iterate through the individual sequence values using foreach. The following runs through the individual values in sequence order.

```
<?php
foreach ($letter->getSequence() as $value) {
```

```
   // ...
}
?>
```

Example #18 - Sequence versus Data Object

Setting values through the data object interface may result in the value not being part of the sequence. A value set through the data object will only be accessible through the sequence if the property was already part of the sequence. The following example sets the lastName through the data object and gets it through the sequence. This is fine because lastName already exists in the sequence. If it had not previously been set, then lastName would be set to 'Smith', but would not be part of the sequence.

```
<?php
 $letter[2] = 'Smith';
 $last_name = $letter_seq[5];
?>
```

Example #19 - Adding to a sequence

We can add new values to a sequence using the SDO_Sequence::insert() method. The following examples assume that the 'firstName' and 'lastName' properties are initially unset.

```
<?php
 // Append a firstName value to the sequence
 // value: 'Smith'
 // sequence index: NULL (append)
 // propertyIdentifier: 1 (firtName property index)
 $letter_seq->insert('Smith', NULL, 1);

 // Append a lastName value to the sequence
 // value: 'Jones'
 // sequence index: NULL (append)
 // propertyIdentifier: 'lastName' (lastName property name)
 $letter_seq->insert('Jones', NULL, 'lastName');

 // Append unstructured text
 // value: 'Cancel Subscription.'
 // sequence index: absent (append)
 // propertyIdentifier: absent (unstructured text)
 $letter_seq->insert('Cancel Subscription.');

 // Insert new unstructured text.  Subsequent sequence values
 // are shifted up.
 // value: 'Care of:'
 // sequence index: 1 (insert as second element)
 // propertyIdentifier: absent (unstructured text)
 $letter_seq->insert('Care of:', 1);
```

```
?>
```

Example #20 - Removing from a sequence

We can use the isset() and unset() functions to test and remove items from the sequence (Note: unset() currently leaves the values in the data object, but this behaviour is likely to change to also remove the data from the data object). A sequence behaves like a contiguous list; therefore, removing items from the middle will shift entries at higher indices down. The following example tests to see if the first sequence element is set and unsets it if is.

```
<?php
 if (isset($letter_seq[0])) {
   unset($letter_seq[0]);
 }
?>
```

Reflecting on Service Data Objects

SDOs have a knowledge of the structure they have been created to represent (the model). For example, a Company SDO created using the Company XML schema above would only be permitted to contain DepartmentType data objects which in turn could only contain EmployeeType data objects.

Sometimes it is useful to be able to access this model information at runtime. For example, this could be used to automatically generate a user interface for populating a data object. The model information is accessed using reflection.

Example #21 - Reflecting on a Data Object

The following example shows how we can reflect on an empty Employee data object.

```
<?php
 // Create the employee data object (e.g. from an XML Data Access Service)
 $employee = ...;
 $reflection = new SDO_Model_ReflectionDataObject($employee);
 print($reflection);
?>
```

The above example will output:

```
object(SDO_Model_ReflectionDataObject)#4 { - ROOT OBJECT - Type {
companyNS:EmployeeType[3] { commonj.sdo:String $name;
commonj.sdo:String $SN; commonj.sdo:Boolean $manager; } }
```

Using print on the SDO_Model_ReflectionDataObject writes out the data object's model. We can see from the output how the type companyNS:EmployeeType has three properties and we can see the names of the properties along with their types. Note, the primitive types are listed as SDO types (e.g. commonj.sdo namespace, String type). It is worth noting that this is the SDO model and when these are surfaced to an application they can be treated as the PHP equivalent types (e.g. string and boolean).

Example #22 - Accessing the type information

We can query the type information of a data object using reflection. The following example checks the type corresponds to a data object rather than a primitive and then iterates through the properties of the type, writing out the name of each property ($type and $property are SDO_Model_Type and SDO_Model_Property objects, respectively).

```
<?php
   // Create the employee data object (e.g. from an XML Data Access Service)
   $employee = ...;
   $reflection = new SDO_Model_ReflectionDataObject($employee);
   $type = $reflection->getType();
   if (! $type->isDataType()) {
       foreach ($type->getProperties() as $property) {
           print $property->getName() . "\n";
       }
   }
?>
```

The above example will output:

```
name
SN
manager
```

SDO Functions

Data Access Services

The table below lists the currently provided SDO Data Access Services:

DAS Name	Description
SDO_DAS_XML	An XML Data Access Service supporting reading/writing SDOs as XML documents.
SDO_DAS_Relational	A PDO-based Data Access Service supporting reading/writing SDO to relational databases. Implements an optimistic concurrency policy for updates.

Predefined Classes

SDO consists of three sets of interfaces. The first set covers those interfaces for use by typical SDO applications. These are identified by the package prefix 'SDO_'. The second set is those used to reflect on, and work with, the model of a data object. These are identified by the package prefix 'SDO_Model_'. Finally, the third set are those use by Data Access Service implementations and are identified by the package prefix 'SDO_DAS_'. The majority of SDO users will not need to use or understand the 'SDO_Model_' and 'SDO_DAS_' interfaces.

SDO Application Programmer Interface

SDO_DataObject

The main interface through which data objects are manipulated. In addition to the methods below, SDO_DataObject extends the ArrayAccess, SDO_PropertyAccess (defines **__get()** / **__set()** methods for property access overloading), Iterator, and Countable interfaces.

Methods

- getSequence - get the sequence for the data object
- createDataObject - create a child data object

- clear - unset the properties of a data object
- getContainer - get the container (also known as 'parent') of this data object
- getTypeName - get the name of the type for this data object
- getTypeNamespaceURI - get the namespace URI of the type for this data object

SDO_Sequence

The interface through which sequenced data objects can be accessed to preserve ordering across a data object's properties and to allow unstructured text. SDO_Sequence preserves contiguous indices and therefore inserting or removing elements may shift other elements up or down. In addition to the methods below, SDO_Sequence extends the ArrayAccess, Iterator and Countable interface.

Methods

- getProperty - get the property for a given sequence index
- move - move an element from one property index to another
- insert - insert a new value into the sequence

SDO_List

The interface through which many-valued properties are manipulated. In addition to the method defined below, SDO_List extends ArrayAccess, Iterator and Countable. SDO_List preserves contiguous indices and therefore inserting or removing elements may shift other elements up or down.

Methods

- insert - insert a new value into the list

SDO_DataFactory

The interface through which data objects can be created. A Data Access Service is responsible for populating the model (i.e. configuring the data factory with the type and structure information for the data objects it can create.) for the factory and can then optionally return an instance of, or implement, the SDO_DataFactory interface.

Methods

- create - create a new data object

SDO_Exception

An SDO_Exception is thrown when the caller's request cannot be completed. The subclasses of SDO_Exception are:

- SDO_PropertyNotSetException - the property specified exists but has not been set or does not have a default value
- SDO_PropertyNotFoundException - the property specified is not part of the data object's type
- SDO_TypeNotFoundException - the specified namespace URI or type name is unknown
- SDO_InvalidConversionException - conversion between the types of the assignment is not possible
- SDO_IndexOutOfBoundsException - the numeric index into a data object, sequence or list is not in the valid range
- SDO_UnsupportedOperationException - the request cannot be completed because it is not allowed, for example an attempt to set a read-only property.

Methods

One method is added to those inherited from the built in Exception class:

- getCause - get the cause of this SDO_Exception

SDO Reflection Application Programmer Interfaces

SDO_Model_ReflectionDataObject

The main interface used to reflect on a data object instance to obtain its model type and property information. It is designed to follow the reflection pattern introduced in PHP 5.

Constructor

- __construct - construct a new SDO_Model_ReflectionDataObject.

Methods

- export - get a string describing the data object.
- getType - get the SDO_Model_Type for the data object.
- getInstanceProperties - get the instance properties of the data object.
- getContainmentProperty - get the property which defines the containment relationship to the data object.

SDO_Model_Type

The interface through which a data object's type information can be retrieved. This interface can be used to find out the type name and namespace URI of the type, whether the type allow open content, and so on.

Methods

- getName - get the name of the type.
- getNamespaceURI - get the namespace URI of the type.
- isInstance - test for a data object being an instance of the type.
- getProperties - get the properties of the type.
- getProperty - get a property of the type.
- isDataType - test to see if this type is a primitive scalar type.
- isSequencedType - test to see if this is a sequenced type.
- isOpenType - test to see if this is an open type.
- isAbstractType - test to see if this is an abstract type.
- getBaseType - get the base type of this type (if one exists).

SDO_Model_Property

The interface through which a data object's property information can be retrieved. This interface can be used to find out the type of a property, whether a property has a default value, whether the property is contained or reference by its parent, its cardinality, and so on.

Methods

- getName - get the name of the property.
- getType - get the type of the property.
- isMany - test to see if the property is many-valued.
- isContainment - test to see if the property describes a containment relationship.
- getContainingType - get the type which contains this property.
- getDefault - get the default value for a property.

SDO Data Access Service Developer Interfaces

SDO_DAS_DataObject

The interface through which a Data Access Service can access a data object's SDO_DAS_ChangeSummary. The change summary is used by the Data Access Service to check for conflicts when applying changes back to a data source.

Methods

- getChangeSummary - get the change summary for a data object

SDO_DAS_ChangeSummary

The interface through which the change history of a data object is accessed. The change summary holds information for any modifications on a data object which occurred since logging was activated. In the case of deletions and modifications, the old values are also held in the change summary.

If logging is no longer active then the change summary only holds changes made up to the point when logging was deactivated. Reactivating logging clears the change summary. This is useful when a set of changes have been written out by a DAS and the data object is to be reused.

Methods

- beginLogging - begin logging changes made to a data object
- endLogging - end logging changes made to a data object
- isLogging - test to see if change logging is on
- getChangedDataObjects - get a list of the data objects which have been changed
- getChangeType - get the type of change which has been made to a data object

- getOldValues - get a list of old values for a data object
- getOldContainer - get the old container data object for a deleted data object

SDO_DAS_Setting

The interface through which the old value for a property is accessed. A list of settings is returned by the change summary method **getOldValues()**.

Methods

- getPropertyIndex - get the property index for the changed property
- getPropertyName - get the property name for the changed property
- getValue - get the old value for the changed property
- getListIndex - get the list index for the old value if it was part of a many-valued property
- isSet - test to see if the property was set prior to being modified

SDO_DAS_DataFactory

The interface for constructing the model for an SDO_DataObject. The SDO_DAS_DataFactory is an abstract class providing a static method which returns a concrete data factory implementation. The implementation is used by Data Access Services to create an SDO model from their model. For example, a Relational Data Access Service might create and populate an SDO_DAS_DataFactory model based on a schema for a relational database.

Methods

- getDataFactory - static methods for getting a concrete data factory instance
- addType - add a new type to the SDO model
- addPropertyToType - add a new property to a type definition in the SDO model

SDO_DAS_ChangeSummary::beginLogging

SDO_DAS_ChangeSummary::beginLogging -- Begin change logging

Description

void **SDO_DAS_ChangeSummary::beginLogging** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Begin logging changes made to the SDO_DataObject.

Parameters

None.

Return Values

None.

SDO_DAS_ChangeSummary::endLogging

SDO_DAS_ChangeSummary::endLogging -- End change logging

Description

void **SDO_DAS_ChangeSummary::endLogging** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

End logging changes made to an SDO_DataObject.

Parameters

None.

Return Values

None.

SDO_DAS_ChangeSummary::getChangeType

SDO_DAS_ChangeSummary::getChangeType -- Get the type of change made to an SDO_DataObject

Description

int **SDO_DAS_ChangeSummary::getChangeType** (SDO_DataObject `$dataObject`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the type of change which has been made to the supplied SDO_DataObject.

Parameters

`dataObject`
The SDO_DataObject which has been changed.

Return Values

The type of change which has been made. The change type is expressed as an enumeration and will be one of the following four values:

- SDO_DAS_ChangeSummary::NONE
- SDO_DAS_ChangeSummary::MODIFICATION
- SDO_DAS_ChangeSummary::ADDITION
- SDO_DAS_ChangeSummary::DELETION

SDO_DAS_ChangeSummary::getChangedDataObjects

SDO_DAS_ChangeSummary::getChangedDataObjects -- Get the changed data objects from a change summary

Description

SDO_List **SDO_DAS_ChangeSummary::getChangedDataObjects** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get an SDO_List of the SDO_DataObjects which have been changed. These data objects can then be used to identify the types of change made to each, along with the old values.

Parameters

None.

Return Values

Returns an SDO_List of SDO_DataObjects.

SDO_DAS_ChangeSummary::getOldContainer

SDO_DAS_ChangeSummary::getOldContainer -- Get the old container for a deleted SDO_DataObject

Description

SDO_DataObject **SDO_DAS_ChangeSummary::getOldContainer** (SDO_DataObject `$data_object`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the old container (SDO_DataObject) for a deleted SDO_DataObject.

Parameters

`data_object`
: The SDO_DataObject which has been deleted and whose container we wish to identify.

Return Values

The old containing data object of the deleted SDO_DataObject.

SDO_DAS_ChangeSummary::getOldValues

SDO_DAS_ChangeSummary::getOldValues -- Get the old values for a given changed SDO_DataObject

Description

SDO_List **SDO_DAS_ChangeSummary::getOldValues** (SDO_DataObject `$data_object`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get a list of the old values for a given changed SDO_DataObject. Returns a list of SDO_DAS_Settings describing the old values for the changed properties of the SDO_DataObject.

Parameters

`data_object`
The data object which has been changed.

Return Values

A list of SDO_DAS_Settings describing the old values for the changed properties of the SDO_DataObject. If the change type is SDO_DAS_ChangeSummary::ADDITION, this list is empty.

SDO_DAS_ChangeSummary::isLogging

SDO_DAS_ChangeSummary::isLogging -- Test to see whether change logging is switched on

Description

bool **SDO_DAS_ChangeSummary::isLogging** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see whether change logging is switched on.

Parameters

None.

Return Values

Returns **TRUE** if change logging is on, otherwise returns **FALSE**.

SDO_DAS_DataFactory::addPropertyToType

SDO_DAS_DataFactory::addPropertyToType -- Adds a property to a type

Description

void **SDO_DAS_DataFactory::addPropertyToType** (string `$parent_type_namespace_uri`, string `$parent_type_name`, string `$property_name`, string `$type_namespace_uri`, string `$type_name` [, array `$options`])

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Adds a property to a type. The type must already be known to the SDO_DAS_DataFactory (i.e. have been added using addType()). The property becomes a property of the type. This is how the graph model for the structure of an SDO_DataObject is built.

Parameters

`parent_type_namespace_uri`
: The namespace URI for the parent type.

`parent_type_name`
: The type name for the parent type.

`property_name`
: The name by which the property will be known in the parent type.

`type_namespace_uri`
: The namespace URI for the type of the property.

`type_name`
: The type name for the type of the property

`options`
: This array holds one or more key=>value pairs to set attribute values for the property. The optional keywords are:

  `many`
  : A flag to say whether the property is many-valued. A value of 'true' adds the property as a many-valued property (default is 'false').

  `readOnly`
  : A flag to say whether the property is read-only. A value of 'true' means the property

value cannot be modified through the SDO application APIs (default is 'false').

containment
: A flag to say whether the property is contained by the parent. A value of 'true' means the property is contained by the parent. A value of 'false' results in a non-containment reference (default is 'true'). This flag is only interpreted when adding properties which are data object types, otherwise it is ignored.

default
: A default value for the property. Omitting this key means that the property does not have a default value. A property can only have a default value if it is a single-valued data type (primitive).

Return Values

None.

ChangeLog

Version	Description
0.5.2	Optional parameters *many*, *readOnly*, and *containment* deprecated in favour of the *options* array.

Examples

Example #23 - A SDO_DAS_DataFactory::addPropertyToType() example

The following adds an 'addressline' property to a Person type. The person type is identified by its namespace, 'PersonNS', and type name, 'PersonType'. The type of the 'addressline' property is a many-valued SDO data type (primitive) with namespace 'commonj.sdo' and type name 'String'.

```
<?php
 $df->addPropertyToType('PersonNS', 'PersonType',
   'addressline', 'commonj.sdo', 'String', array('many'=>true));
?>
```

SDO_DAS_DataFactory::addType

SDO_DAS_DataFactory::addType -- Add a new type to a model

Description

void **SDO_DAS_DataFactory::addType** (string `$type_namespace_uri`, string `$type_name` [, array `$options`])

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Add a new type to the SDO_DAS_DataFactory, defined by its namespace and type name. The type becomes part of the model of data objects that the data factory can create.

Parameters

`type_namespace_uri`
: The namespace of the type.

`type_name`
: The name of the type.

`options`
: This array holds one or more key=>value pairs to set attribute values for the type. The optional keywords are:

  `open`
  : A flag to say whether the type is open. An SDO_DataObject whose type is open can have properties added to them which are not described by the type. This capability is used to support working with XML documents whose schema support open content such as that described by an <xsd:any> element. The default value is 'false'.

  `sequenced`
  : A flag to say whether the type is sequenced. Sequenced types can have the ordering across properties preserved and can contain unstructured text. The default value is 'false'. For more information on sequenced types see the section on Working with Sequenced Data Objects.

  `basetype`
  : If specified, an array of namespace URI and type name strings for the type from which this type is derived. An example of the use of base types is when a type derived in an XML schema inherits from another type by using <extension

base="...">.

Return Values

None.

Examples

Example #24 - A SDO_DAS_DataFactory::addType() example

The following adds a new data object type of 'CompanyType' where that type belongs to the namespace 'CompanyNS'.

```
<?php
 $df->addType('CompanyNS', 'CompanyType');
?>
```

SDO_DAS_DataFactory::getDataFactory

SDO_DAS_DataFactory::getDataFactory -- Get a data factory instance

Description

SDO_DAS_DataFactory **SDO_DAS_DataFactory::getDataFactory** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Static method to get an instance of an SDO_DAS_DataFactory. This instance is initially only configured with the basic SDO types. A Data Access Service is responsible for populating the data factory model and then allowing PHP applications to create SDOs based on the model through the SDO_DataFactory interface. PHP applications should always obtain a data factory from a configured Data Access Service, not through this interface.

Parameters

None.

Return Values

Returns an SDO_DAS_DataFactory.

SDO_DAS_DataObject::getChangeSummary

SDO_DAS_DataObject::getChangeSummary -- Get a data object's change summary

Description

SDO_DAS_ChangeSummary **SDO_DAS_DataObject::getChangeSummary** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the SDO_DAS_ChangeSummary for an SDO_DAS_DataObject, or NULL if it does not have one.

Parameters

None.

Return Values

Returns the SDO_DAS_ChangeSummary for an SDO_DAS_DataObject, or NULL if it does not have one.

SDO_DAS_Setting::getListIndex

SDO_DAS_Setting::getListIndex -- Get the list index for a changed many-valued property

Description

int **SDO_DAS_Setting::getListIndex** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the list index for a modification made to an element of a many-valued property. For example, if we modified the third element of a many-valued property we could obtain an SDO_DAS_Setting from the change summary corresponding to that modification. A call to **getListIndex()** on that setting would return the value 2 (lists are indexed from zero).

Parameters

None.

Return Values

The list index for the element of the many-valued property which has been changed.

SDO_DAS_Setting::getPropertyIndex

SDO_DAS_Setting::getPropertyIndex -- Get the property index for a changed property

Description

int **SDO_DAS_Setting::getPropertyIndex** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the property index for the changed property. This index identifies the property which was modified in data object.

Parameters

None.

Return Values

The property index for a changed property.

SDO_DAS_Setting::getPropertyName

SDO_DAS_Setting::getPropertyName -- Get the property name for a changed property

Description

string **SDO_DAS_Setting::getPropertyName** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the property name for the changed property. This name identifies the property which was modified in data object.

Parameters

None.

Return Values

The property name for a changed property.

SDO_DAS_Setting::getValue

SDO_DAS_Setting::getValue -- Get the old value for the changed property

Description

mixed **SDO_DAS_Setting::getValue** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the old value for the changed property. This can be used by a Data Access Service when writing updates to a data source. The DAS uses the old value to detect conflicts by comparing it with the current value in the data source. If they do not match, then the data source has been updated since the data object was originally populated, and therefore writing any new updates risks compromising the integrity of the data.

Parameters

None.

Return Values

Returns the old value of the changed property.

SDO_DAS_Setting::isSet

SDO_DAS_Setting::isSet -- Test whether a property was set prior to being modified

Description

bool **SDO_DAS_Setting::isSet** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test whether a property was set prior to being modified. If it was set prior to being modified then the SDO_DAS_Setting will also contain the old value.

Parameters

None.

Return Values

Returns **TRUE** if the property was set prior to being modified, otherwise returns **FALSE**.

SDO_DataFactory::create

SDO_DataFactory::create -- Create an SDO_DataObject

Description

void **SDO_DataFactory::create** (string `$type_namespace_uri`, string `$type_name`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Create a new SDO_DataObject given the data object's namespace URI and type name.

Parameters

`type_namespace_uri`
The namespace of the type.

`type_name`
The name of the type.

Return Values

Returns the newly created SDO_DataObject.

Errors/Exceptions

`SDO_TypeNotFoundException`
Thrown if the namespaceURI and typeName do not correspond to a type known to this data factory.

SDO_DataObject::clear

SDO_DataObject::clear -- Clear an SDO_DataObject's properties

Description

void **SDO_DataObject::clear** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Clear an SDO_DataObject's properties. Read-only properties are unaffected. Subsequent calls to isset() for the data object will return **FALSE**.

Parameters

None.

Return Values

No return values.

SDO_DataObject::createDataObject

SDO_DataObject::createDataObject -- Create a child SDO_DataObject

Description

SDO_DataObject **SDO_DataObject::createDataObject** (mixed `$identifier`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Create a child SDO_DataObject of the default type for the property identified. The data object is automatically inserted into the tree and a reference to it is returned.

Parameters

`identifier`
: Identifies the property for the data object type to be created. Can be either a property name (string), a property index (int), or an SDO_Model_Property.

Return Values

Returns the newly created SDO_DataObject.

Errors/Exceptions

`SDO_PropertyNotFoundException`
: Thrown if the identifier does not correspond to a property of the data object.

SDO_DataObject::getContainer

SDO_DataObject::getContainer -- Get a data object's container

Description

SDO_DataObject **SDO_DataObject::getContainer** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the data object which contains this data object.

Parameters

None.

Return Values

Returns the SDO_DataObject which contains this SDO_DataObject, or returns NULL if this is a root SDO_DataObject (i.e. it has no container).

SDO_DataObject::getSequence

SDO_DataObject::getSequence -- Get the sequence for a data object

Description

SDO_Sequence **SDO_DataObject::getSequence** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Return the SDO_Sequence for this SDO_DataObject. Accessing the SDO_DataObject through the SDO_Sequence interface acts on the same SDO_DataObject instance data, but preserves ordering across properties.

Parameters

None.

Return Values

The SDO_Sequence for this SDO_DataObject, or returns NULL if the SDO_DataObject is not of a type which can have a sequence.

SDO_DataObject::getTypeName

SDO_DataObject::getTypeName -- Return the name of the type for a data object.

Description

string **SDO_DataObject::getTypeName** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Return the name of the type for a data object. A convenience method corresponding to SDO_Model_ReflectionDataObject::getType().getName().

Parameters

None.

Return Values

The name of the type for the data object.

SDO_DataObject::getTypeNamespaceURI

SDO_DataObject::getTypeNamespaceURI -- Return the namespace URI of the type for a data object.

Description

string **SDO_DataObject::getTypeNamespaceURI** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Return the namespace URI of the type for a data object. A convenience method corresponding to SDO_Model_ReflectionDataObject::getType().getNamespaceURI().

Parameters

None.

Return Values

The namespace URI of the type for the data object.

SDO_Exception::getCause

SDO_Exception::getCause -- Get the cause of the exception.

Description

mixed **SDO_Exception::getCause** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the cause of this exception or NULL if the cause is nonexistent or unknown. Typically the cause will be an SDO_CPPException object, which may be used to obtain additional diagnostic information.

Parameters

None.

Return Values

Returns the cause of this exception or NULL if the cause is nonexistent or unknown.

SDO_List::insert

SDO_List::insert -- Insert into a list

Description

void **SDO_List::insert** (mixed `$value` [, int `$index`])

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Insert a new element at a specified position in the list. All subsequent list items are moved up.

Parameters

`value`
The new value to be inserted. This can be either a primitive or an SDO_DataObject.

`index`
The position at which to insert the new element. If this argument is not specified then the new value will be appended.

Return Values

None.

Errors/Exceptions

`SDO_IndexOutOfBoundsException`
Thrown if the list index is less than zero or greater than the size of the list.

`SDO_InvalidConversionException`
Thrown if the type of the new value does not match the type for the list (e.g. the type of the many-valued property that the list represents).

SDO_Model_Property::getContainingType

SDO_Model_Property::getContainingType -- Get the SDO_Model_Type which contains this property

Description

SDO_Model_Type **SDO_Model_Property::getContainingType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the SDO_Model_Type which contains this property.

Parameters

None.

Return Values

Returns the SDO_Model_Type which contains this property.

SDO_Model_Property::getDefault

SDO_Model_Property::getDefault -- Get the default value for the property

Description

mixed **SDO_Model_Property::getDefault** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the default value for the property. Only primitive data type properties can have default values.

Parameters

None.

Return Values

Returns the default value for the property.

SDO_Model_Property::getName

SDO_Model_Property::getName -- Get the name of the SDO_Model_Property

Description

string **SDO_Model_Property::getName** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the name of the SDO_Model_Property.

Parameters

None.

Return Values

Returns the name of the SDO_Model_Property.

SDO_Model_Property::getType

SDO_Model_Property::getType -- Get the SDO_Model_Type of the property

Description

SDO_Model_Type **SDO_Model_Property::getType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the SDO_Model_Type of the property. The SDO_Model_Type describes the type information for the property, such as its type name, namespace URI, whether it is a primitive data type, and so on.

Parameters

None.

Return Values

Returns the SDO_Model_Type describing the property's type information.

SDO_Model_Property::isContainment

SDO_Model_Property::isContainment -- Test to see if the property defines a containment relationship

Description

bool **SDO_Model_Property::isContainment** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see if the property corresponds to a containment relationship. Returns **TRUE** if the property defines a containment relationship, or **FALSE** if it is reference.

Parameters

None.

Return Values

Returns **TRUE** if the property defines a containment relationship, or **FALSE** if it is reference.

SDO_Model_Property::isMany

SDO_Model_Property::isMany -- Test to see if the property is many-valued

Description

bool **SDO_Model_Property::isMany** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see if the property is many-valued. Returns **TRUE** if this is a many-valued property, otherwise returns **FALSE**.

Parameters

None.

Return Values

Returns **TRUE** if this is a many-valued property, otherwise returns **FALSE**.

SDO_Model_ReflectionDataObject::__construct

SDO_Model_ReflectionDataObject::__construct -- Construct an SDO_Model_ReflectionDataObject

Description

SDO_Model_ReflectionDataObject **SDO_Model_ReflectionDataObject::__construct** (SDO_DataObject `$data_object`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Construct an SDO_Model_ReflectionDataObject to reflect on an SDO_DataObject. Reflecting on an SDO_DataObject gives access to information about its model. The model contains information such as the data object's type, and whether that type is sequenced (preserves ordering across properties) or open (each instance can have its model extended). The model also holds information about the data object's properties, any default values they may have, and so on.

Parameters

`data_object`
The SDO_DataObject being reflected upon.

Return Values

None.

SDO_Model_ReflectionDataObject::export

SDO_Model_ReflectionDataObject::export -- Get a string describing the SDO_DataObject.

Description

mixed **SDO_Model_ReflectionDataObject::export** (SDO_Model_ReflectionDataObject `$rdo` [, bool `$return`])

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get a string describing the SDO_DataObject. The default behaviour is to print the output, but if **TRUE** is specified for return, it is returned as a string.

Parameters

`rdo`
An SDO_Model_ReflectionDataObject.

`return`
If **TRUE**, return the output as a string, otherwise print it.

Return Values

Returns the output if TRUE is specified for return, otherwise NULL.

SDO_Model_ReflectionDataObject::getContainmentProperty

SDO_Model_ReflectionDataObject::getContainmentProperty -- Get the property which defines the containment relationship to the data object

Description

SDO_Model_Property **SDO_Model_ReflectionDataObject::getContainmentProperty** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the SDO_Model_Property that contains the SDO_DataObject. This method is used to navigate up to the parent's property which contains the data object which has been reflected upon.

Parameters

None.

Return Values

Returns the container's SDO_Model_Property which references the SDO_DataObject, or **NULL** if it is a root SDO_DataObject.

SDO_Model_ReflectionDataObject::getInstanceProperties

SDO_Model_ReflectionDataObject::getInstanceProperties -- Get the instance properties of the SDO_DataObject

Description

array **SDO_Model_ReflectionDataObject::getInstanceProperties** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the instance properties for the SDO_DataObject. The instance properties consist of all the properties defined on the data object's type, plus any instance properties from open content (if the data object is an open type).

Parameters

None.

Return Values

An array of SDO_Model_Property objects.

SDO_Model_ReflectionDataObject::getType

SDO_Model_ReflectionDataObject::getType -- Get the SDO_Model_Type for the SDO_DataObject

Description

SDO_Model_Type **SDO_Model_ReflectionDataObject::getType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the SDO_Model_Type for the SDO_DataObject. The SDO_Model_Type holds all the information about the data object's type, such as namespace URI, type name, whether it is a primitive data type, and so on.

Parameters

None.

Return Values

Returns the SDO_Model_Type for the SDO_DataObject.

SDO_Model_Type::getBaseType

SDO_Model_Type::getBaseType -- Get the base type for this type

Description

SDO_Model_Type **SDO_Model_Type::getBaseType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get the base type for this type. Returns the SDO_Model_Type for the base type if this type inherits from another, otherwise returns **NULL**. An example of when base types occur is when a type defined in XML schema inherits from another type by using<extension base="...">.

Parameters

None.

Return Values

Returns the SDO_Model_Type for the base type if this type inherits from another, otherwise returns **NULL**.

SDO_Model_Type::getName

SDO_Model_Type::getName -- Get the name of the type

Description

string **SDO_Model_Type::getName** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the name of the type. The combination of type name and namespace URI is used to uniquely identify the type.

Parameters

None.

Return Values

Returns the name of the type.

SDO_Model_Type::getNamespaceURI

SDO_Model_Type::getNamespaceURI -- Get the namespace URI of the type

Description

string **SDO_Model_Type::getNamespaceURI** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Returns the namespace URI of the type. The combination of namespace URI and type name is used to uniquely identify the type.

Parameters

None.

Return Values

Returns the namespace URI of the type.

SDO_Model_Type::getProperties

SDO_Model_Type::getProperties -- Get the SDO_Model_Property objects defined for the type

Description

array **SDO_Model_Type::getProperties** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get an array of SDO_Model_Property objects describing the properties defined for the SDO_Model_Type. Each SDO_Model_Property holds information such as the property name, default value, and so on.

Parameters

None.

Return Values

Returns an array of SDO_Model_Property objects.

SDO_Model_Type::getProperty

SDO_Model_Type::getProperty -- Get an SDO_Model_Property of the type

Description

SDO_Model_Property **SDO_Model_Type::getProperty** (mixed `$identifier`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Get an SDO_Model_Property of the type, identified by its property index or property name.

Parameters

`identifier`
The property index or property name.

Return Values

Returns the SDO_Model_Property.

SDO_Model_Type::isAbstractType

SDO_Model_Type::isAbstractType -- Test to see if this SDO_Model_Type is an abstract data type

Description

bool **SDO_Model_Type::isAbstractType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see if this SDO_Model_Type is an abstract data type. Returns **TRUE** if this type is abstract, that is, no SDO_DataObject of this type can be instantiated, though other types may inherit from it.

Parameters

None.

Return Values

Returns **TRUE** if this type is an abstract data type, otherwise returns **FALSE**.

SDO_Model_Type::isDataType

SDO_Model_Type::isDataType -- Test to see if this SDO_Model_Type is a primitive data type

Description

bool **SDO_Model_Type::isDataType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see if this SDO_Model_Type is a primitive data type. Returns **TRUE** if this type is a primitive data type, otherwise returns **FALSE**.

Parameters

None.

Return Values

Returns **TRUE** if this type is a primitive data type, otherwise returns **FALSE**.

SDO_Model_Type::isInstance

SDO_Model_Type::isInstance -- Test for an SDO_DataObject being an instance of this SDO_Model_Type

Description

bool **SDO_Model_Type::isInstance** (SDO_DataObject `$data_object`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test for an SDO_DataObject being an instance of this SDO_Model_Type. Returns **TRUE** if the SDO_DataObject provided is an instance of this SDO_Model_Type, or a derived type, otherwise returns **FALSE**.

Parameters

`data_object`
The SDO_DataObject to be tested.

Return Values

Returns **TRUE** if the SDO_DataObject provided is an instance of this SDO_Model_Type, or a derived type, otherwise returns **FALSE**.

SDO_Model_Type::isOpenType

SDO_Model_Type::isOpenType -- Test to see if this type is an open type

Description

bool **SDO_Model_Type::isOpenType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see if this type is open. Returns **TRUE** if this type is open, otherwise returns **FALSE**. An SDO_DataObject whose type is open can have properties added to them which are not described by the type. This capability is used to support working with XML documents whose schema support open content, such as that defined by an<xsd:any> element.

Parameters

None.

Return Values

Returns **TRUE** if this type is open, otherwise returns **FALSE**.

SDO_Model_Type::isSequencedType

SDO_Model_Type::isSequencedType -- Test to see if this is a sequenced type

Description

bool **SDO_Model_Type::isSequencedType** (void)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Test to see if this is a sequenced type. Returns **TRUE** if this type is sequence, otherwise returns **FALSE**. Sequenced types can have the ordering across properties preserved and can contain unstructured text. For more information on sequenced types see the section on Working with Sequenced Data Objects.

Parameters

None.

Return Values

Returns **TRUE** if this type is sequence, otherwise return **FALSE**.

SDO_Sequence::getProperty

SDO_Sequence::getProperty -- Return the property for the specified sequence index.

Description

SDO_Model_Property **SDO_Sequence::getProperty** (int `$sequence_index`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Return the property for the specified sequence index.

Parameters

`sequence_index`
: The position of the element in the sequence.

Return Values

An SDO_Model_Property. A return value of NULL means the sequence element does not belong to a property and must therefore be unstructured text.

Errors/Exceptions

`SDO_IndexOutOfBoundsException`
: Thrown if the sequence index is less than zero or greater than the size of the sequence.

SDO_Sequence::insert

SDO_Sequence::insert -- Insert into a sequence

Description

void **SDO_Sequence::insert** (mixed `$value` [, int `$sequenceIndex` [, mixed `$propertyIdentifier`]])

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Insert a new element at a specified position in the sequence. All subsequent sequence items are moved up.

Parameters

`value`
: The new value to be inserted. This can be either a primitive or an SDO_DataObject.

`sequenceIndex`
: The position at which to insert the new element. Default is NULL, which results in the new value being appended to the sequence.

`propertyIdentifier`
: Either a property index, a property name, or an SDO_Model_Property, used to identify a property in the sequence's corresponding SDO_DataObject. A value of NULL signifies unstructured text.

Return Values

None.

Errors/Exceptions

`SDO_IndexOutOfBoundsException`
: Thrown if the sequence index is less than zero or greater than the size of the sequence.

`SDO_InvalidConversionException`
: Thrown if the type of the new value cannot be juggled to match the type for the

specified data object property.

SDO_Sequence::move

SDO_Sequence::move -- Move an item to another sequence position

Description

void **SDO_Sequence::move** (int `$toIndex`, int `$fromIndex`)

Warning
This function is *EXPERIMENTAL*. The behaviour of this function, its name, and surrounding documentation may change without notice in a future release of PHP. This function should be used at your own risk.

Modify the position of the item in the sequence, without altering the value of the property in the SDO_DataObject.

Parameters

`toIndex`
: The destination sequence index. If this index is less than zero or greater than the size of the sequence then the value is appended.

`fromIndex`
: The source sequence index.

Return Values

None.

Errors/Exceptions

`SDO_IndexOutOfBoundsException`
: Thrown if the fromIndex sequence index is less than zero or greater than the size of the sequence.