Migrating UniData ObjectCall to UniObjects

Introduction

Release 5.1 of UniData was the "common middleware" release incorporating existing and new Informix technology for middleware solutions. As part of this release we introduced InterCall, UniObjects and UniObjects for Java to UniData users. UniData 5.1 also included UniOLEDB, middleware supporting Microsoft's API for universal data access to both SQL and non-SQL data sources.

InterCall and the UniObjects APIs provide high-performance, native APIs to both UniData and UniVerse. InterCall and UniObjects provide access to UniData similar to that provided by ObjectCall and ObjectCall/EasyX respectively. UniObjects for Java provides new Java-based API into UniData as well as UniVerse.

Please note that, in release 5.1 and going forward, ObjectCall will still be available to UniData users. This was done expressly to provide an easier migration path by allowing existing applications to continue functioning while developers work on migrating to the newer UniObjects technology. However, ObjectCall is a deprecated product that will not be enhanced or repaired. Since implementation of device licensing and on-going enhancements will be made only in the InterCall/UniObjects products, this document provides a guide to the migration effort.

Summary

This document provides technical guidance on issues arising from the migration of an application using ObjectCall to one using UniObjects. Although these two products are very similar in functionality, they are at different levels of abstraction. ObjectCall is a C interface API and UniObjects is a COM-based custom control. ObjectCall is actually at the same level of abstraction as InterCall and UniObjects is at the same level as ObjectCall/EasyX.

Migrating from ObjectCall/EasyX to UniObjects should be very straightforward as they are at the same level of abstraction and EasyX was expressly designed to emulate UniObjects. However, some architectural differences remain.

This documents only looks at the ObjectCall to UniObjects differences. Since this document describes the migration of ObjectCall to UniObjects, additional functionality that exists in UniObjects is not covered. For complete information on UniObjects see the "UniObjects Developer's Guide" on-line documentation.

In working through a function-for-function and property-for-property mapping there is very little that is not covered. The number of possible wrapper functions to complete this mapping further would be fairly small. Each wrapper will probably require a bit of overhead since it would emulate behavior existing in a single function or property.

This document assumes some familiarity with the Informix extended relational database object-based middleware products. For complete information on ObjectCall, ObjectCall with EasyX, InterCall, and UniObjects see the UniData 5.1 on-line documentation.

Major Differences

This section discusses the major architectural differences between ObjectCall and UniObjects. Each area discusses a potential workaround.

Aggregates

The most significant conceptual difference between UniObjects and ObjectCall is the use of aggregates to represent record data. This can be a powerful way to manipulate nested data of any level and is extensively used within ObjectCall. However, aggregates tend to be complicated to understand and use.

UniObjects uses a DynamicArray object that functionally performs much the same, although it only supports 3 nesting levels, equivalent to single, multi-, and multi-sub values. The UniObjects DynamicArray object is not used as fundamentally as ObjectCall uses the aggregate.

Record Access

The next most significant conceptual difference between UniObjects and ObjectCall is in accessing and updating records. Most ObjectCall applications open a table, then open a row and perform operations on that row as a persistent object. There is no need to explicitly write the data back to the database. Closing the row automatically saves any changes made.

UniObjects requires any record read from the table to be written back after modifications have been done. Practically speaking, this is only a matter of replacing the ObjectCall row Open and Close functions with the UniObjects table Read and Write functions.

Dictionary Filters

ObjectCall allows the user to filter the record read in the server so that only certain fields of interest are returned to the client. UniObjects allows you to access fields in a record one at a time but not multiple at once.

Multiple Reads and Writes

ObjectCall's UniMultiRead and UniMultiWrite functions improve performance by doing large numbers of reads and writes. UniObjects doesn't support these functions directly but can do the MultiRead by creating a select list then using the ReadList function. A subroutine will be needed to perform the MultiWrite functionality. **Note: UniObjects for Java includes the UniDataSet object that performs a function similar to UniMultiRead.**

Record Locks

ObjectCall locks records by opening and manipulating a Lock object on a table. UniObjects locks records by calling the LockRecord function.

Remote Executions

ObjectCall is able to provide a list of commands to be performed by the server; UniObjects performs commands one at a time. Only a few customers are known to use this ObjectCall feature.

A wrapper function could be written in UniObjects to take a list of commands and execute them one by one.

The UNItermIn property passes terminal input to commands run by ObjectCall using the UniRun function. In UniObjects, the Command.Reply function supplies this input interactively.

Possible wrapper functions

The following table describes the wrapper functions that need to be created for UniObjects to provide the equivalent ObjectCall functionality:

Differing ObjectCall Functionality	**UniObjects Solution**
Multiple Writes (MultiWrite)	Create and call a UniBasic subroutine on the host
Remote Executions	Create a wrapper function to repeatedly call Command.Exec
Dictionary Filters	For small numbers of fields, use the UniObjects ReadField command. For large numbers of fields, use the Read command to read the whole record and remove unwanted fields.
Table Count	Create and call a UniBasic subroutine that counts records.

Function Mappings

This section lists all of the ObjectCall functions and describes how to achieve the same or similar functionality using UniObjects.

UniAgrFromStr

UniObjects's DynamicArray object maps fairly well to the Aggregate object and the default StringValue property of the DynamicArray performs this operation internally when assigned a value.

UniCleanUp

In ObjectCall, this function closes all open objects, writes back any unwritten data to the database, closes the database connection and cleans up any internal tables and memory used by ObjectCall. No other ObjectCall function call can be made after this call unless UniStartUp is called again.

UniObjects doesn’t have this global cleanup operation but the Session object's Disconnect method will close any open files and locks and will release the session. Setting the Session object to nothing (in Visual Basic) cleans up any memory associated with it.

UniClose

The UniClose function closes various ObjectCall objects. The UniObjects mapping for UniClose depends on what type of object is being closed by UniClose:

ObjectCall Object	UniObjects Mapping
UNIserver	Session.Disconnect
UNIdatabase	Session.Disconnect (Database concept not supported)
UNItable	File.CloseFile
UNIrex	N/A. UniObjects has one Command object held by Session object
UNIrpc	N/A UniObjects uses a Subroutine method on Session object
UNIrow	ObjectCall writes data back to database on close so UniObjects should call FileWrite method before setting DynamicArray variable to nothing
UNIlock	File.UnlockRecord
UNIagr	N/A. DynamicArrays are cleaned up when they go out of scope
UNIstr	N/A. String objects not used

UniCopy

The UniCopy function is used to break the persistent nature of some pieces of data. You can take a copy of the object (UNIagr, UNIrow or UNIstr) and modify the copy and not affect the original persistent version. All data read from the database in UniObjects is non-persistent until written back so data objects can be copied by assignment.

UniCreate

The creation of UniObjects COM objects is specific to the development environment. For example, in Visual Basic you would use the CreateObject function. Almost all objects are derived from other objects in UniObjects except for the Session object and temporary DynamicArray objects.

UniDelDict

N/A. Dictionary filters are not supported in UniObjects.

UniDelete

File.DeleteRecord

UniDelPos

DynamicArray.[Context].Del. A Position value of 0 (all members) in ObjectCall can be achieved by not specifying a Context value, but using –1 (the last member) is not supported.

UniDestroy

File.DeleteRecord. UniDestroy normally operates with a row handle but the DeleteRecord function will need the row name.

UniGetClass

N/A. No Object handles in UniObjects.

UniGetCnt

DynamicArray.Count maps to UniGetCnt on UNIrow objects but there is no mapping on UNIserver, UNIdatabase or UNItable objects.

UniGetEventCode

Each UniObjects object has an Error property that contains the error code for the last operation.

UniGetLabel

N/A. UniGetLabel returns a text description for each of the internal constants.

UniGenLen

Use DynamicArray.Length or a development platform library function that returns the string length. For example, in Visual Basic you would use Len().

UniGetLevel

N/A. UniObjects error codes are grouped in various numerical ranges but there is not a notion of levels of severity. UniObjects separates bad states into Exceptions and Errors with the Exceptions being Fatal Errors.

UniGetNull

In UniObjects you can test if a string is the NULL value by comparing it with the Session.GetAtVariable(AT_NULLSTR) value.

UniGetParent

N/A.

UniGetProp

UniObjects properties are handled with the "dot" notation, i.e., Session.Hostname

UniGetRtn

Use object.Error to get last error code.

UniGetStatus

There is no status object in UniObjects but the same thing can be implemented using a global Exception handling function and setting the ExceptionOnError property to True for all objects. Each object in UniObjects has a Status property, which contains the status of the object (usually the status of the last operation).

UniGetValInt

Use Field, Value and SubValue methods of DynamicArray object to extract data.

UniGetValStr

Use Field, Value and SubValue methods of DynamicArray object to extract data.

UniInqProp

N/A.

UniInsert

File.Write. Writing to a new record ID inserts it.

UniInsPos

DynamicArray.Ins. Insert position of –1 (append at the end) is not supported. Use Count to determine the number of entries then Ins at a position of Count plus one.

UniInterrupt

Command.Cancel for REX calls. Not supported on RPC (SubRoutine) calls.

UniMRead

Use the SelectList object to do Multi Reads. Call the FormList method to create a SelectList on the server the call ReadList to bring back the data.

UniMWrite

Need to write a subroutine on the server that accepts the record ID's and data.

UniOpen

The UniObjects mapping for the UniOpen function depends on the object being opened:

ObjectCall Object	UniObjects Mapping
UNIdatabase	Session.Connect
UNItable	Session.OpenFile
UNIrow	File.Read (This is non-persistent)

UniOpenLock

UNIlock	File.LockRecord

UniOpenPos

Connections, files and records in UniObjects can only be opened by name so the mappings are the same as for the ObjectCall UniOpen function. For opening a position in a record, UniOpenPos maps to the Field, Value and SubValue methods of the DynamicArray object.

UniRead

File.Read

UniReadBck

Cannot read backwards on an Alternate key.

UniReadFwd

Use the SelectList object and the Next method in combination with File.Read. To read multiple records you would have to create another SelectList with the ID's returned by Next then use ReadList to bring them back.

UniReadString

File.Read

UniReplace

File.Write. Writing to an existing record replaces it.

UniRepPos

DynamicArray.Replace. Setting a Replace position of –1 (the last member) is not supported. Use Count to determine the number of entries then Replace at that position.

UniRun

Command.Exec or Subroutine.Call

UniRunRex

Command.Exec. Use the Command.Text property to set the command to run. ECLTYPE cannot be selected at runtime; this is set up at account creation time. One synchronous command can be run at a time. Use Command.Reply to supply input data.

UniRunRpc

Subroutine.Call. The Session.Subroutine call is passed the routine name to run and creates a Subroutine object. Use the SetArg method to set up arguments.

UniSetDictInt

ObjectCall uses this function to set up a dictionary filter for all subsequent reads and writes. UniObjects uses File.ReadField and File.WriteField to read or write a single field immediately.

UniSetDictStr

Same as UniSetDictInt except UniObjects uses the functions File.ReadNamedField and File.WriteNamedField. Note that the XxxxNamedField function always do the input or output conversion as if UniSetDictStrCnv had been set. If you don't want the conversion then use the XxxxField function.

UniSetDictStrCnv

The conversion is always applied when using File.XxxxNamedField functions and not applied when calling File.XxxxField functions.

UniSetIndex

SelectList.SelectAlternateKey

UniSetNull

Set the DynamicArray Field, Value, SubValue method to the Session.GetAtVariable(AT_NULLSTR) value.

UniSetParent

N/A

UniSetValInt

Use Field, Value and SubValue methods of DynamicArray object to set data.

UniSetValStr

Use the Field, Value and SubValue methods of DynamicArray object to set data.

UniStartUp

Creating the initial Session object in UniObjects effectively does the same thing.

UniStatusHandler

UniObjects does not have a StatusHandler function like ObjectCall does but the same thing can be achieved using Exception Handlers.

UniStrFromAgr

The DynamicArray object maps fairly well to the Aggregate object and the default StringValue property of the DynamicArray performs this operation when queried.

UniWait

UniObjects always waits for it's Command and Subroutine calls to finish before returning. The data generated from these can be read back incrementally though through Command.Response and Command.NextBlock.

UniWriteString

File.Write

Property Mappings

The following section describes how to map ObjectCall properties to UniObjects properties:

UNIargs

Subroutine.SetArg or Subroutine.ResetArgs

UNIchildClass

N/A

UNIchildNames

This is not supported at the UNIclient, UNIServer, and UNIdatabase level but the names of the records in a table can be retrieved using a SelectList object.

UNIclass

N/A

UNIcmds

Command.Text. This only takes a single command at a time, however a wrapper function could be written to take a list of commands and execute them one at a time.

UNIcount

Same as UniGetCount function.

UNIdatabaseName

Session.AccountPath

UNIdict

UniObjects uses a separate object to handle dictionary files. Calling Session.OpenDictionary creates the Dictionary object.

UNIdictionary

This aggregate contains all the dictionary filters set up by calls to UniSetDictInt or UniSetDictStr. UniObjects has no equivalent.

UNIdictChild

See UNIdict.

UNIdone

N/A. UniObjects Command and Subroutine calls return when they are done.

UNIeventCode

See UniGetEventCode.

UNIeventLevel

See UniGetLevel.

UNIeventRemote

N/A. UniObjects error codes don't distinguish between local and remote origin.

UNIfmValue

Session.FM

UNIfunc

This is supported by the development environment exception handling rather than by UniObjects itself.

UNIhostNames

N/A UniObjects does not have a lookup list of possible servers.

UNIinternetAddress

Session.HostName

UNIlength

The mechanism for determining the length of the screen in UniObjects is development environment specific. For example, in Visual Basic you would use the Len() function.

UNIlockType

LockType is passed as a parameter to File.RecordLock or is obtained from the Session.DefaultLockStrategy or File.LockStrategy properties.

UNIlockTypeChild

See UNIlockType above.

UNIlockWait

UniObjects doesn't have a specific lock object so if RecordLock succeeds you cannot tell if it had to wait for the lock.

UNIlockWaitChild

File.LockStrategy

UNIlogDir

ObjectCall has the ability to log events on both the client and server machines to assist in diagnosing problems. UniObjects does not support logging.

UNIlogLevel

UniObjects does not support logging.

UNImodified

N/A Only useful when row is persistent.

UNImsg

Only error codes are available.

UNIname

N/A

UNInull

See UniGetNull function

UNIoconv

See UNIdictionary.

UNIpacketSize

This is handled by the UniObjects RPC layer and is not configurable.

UNIparent

N/A

UNIpasswordChild

Session.Password

UNIport

Session.HostName

UNIreadOnly

Read-Only mode is not supported in UniObjects.

UNIreadOnlyChild

Same as UNIreadOnly

UNIreopenChild

UniObjects always reopens files.

UNIresponseSize

See UNIpacketSize and also Command.BlockSize

UNIresultsCodes

Command.CommandStatus one command at a time.

UNIresultsSets

SelectList.GetList

UNIrexType

N/A. This property sets the ECLTYPE for a given command. With UniObjects you must set this up in the account itself.

UNIrowPickFormat

DynamicArray.StringValue

UNIrpcType

N/A

UNIserverName

N/A

UNIstatusHandle

N/A

UNIstatusLevel

N/A

UNIstatusLevelChild

N/A

UNIsync

Both Command and Subroutine are always synchronous.

UNItermIn

Command.Reply supplies input data as requested.

UNItermOut

Command.Response and Command.Reply.

UNIuserName

Session.UserName

UNIuserNameChild

Session.UserName

UNIvalue

DynamicArray Field, Value and SubValue methods.