AUTHORIZED DOCUMENTATION

UCX SDK Components

Novell® Developer Kit

June 21, 2006

www.novell.com

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. Please refer to www.novell.com/info/exports/ for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc. has intellectual property rights relating to technology embodied in the product that is described in this document. In particular, and without limitation, these intellectual property rights may include one or more of the U.S. patents listed at http://www.novell.com/company/legal/patents/ and one or more additional patents or pending patent applications in the U.S. and in other countries.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the online documentation for this and other Novell products, and to get updates, see www.novell.com/documentation.

Novell Trademarks

For a list of Novell Trademarks, see Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html)

Contents

UCX SDK Components

Universal Component eXtension or an UCX component is an abstract unit of functionality. It consists of properties, methods and events. These components are reusable and can be used from scripting languages to develop applications. These components can also be used within the applications developed using high-high-level languages such as C/C++ and Java.

Concepts

1

UCX component is a NetWare Loadable Module (NLM) library. UCX Service, Novell's NetWare SDK (http://developer.novell.com/ndk/nwsdkc.htm) and a C/C++ compiler are required to develop UCX components. UCX service consists of a UCX library manager, an Application Programming Interface (API) and development guidelines. Various functions of UCX API help in creation and usage of UCX components.

The UCX components exhibit the following features

- Abstraction: UCX components are abstract unit of functionality. It consists of properties, methods and events, which expose various services.
- Reusable: These components can be accessed within applications developed using scripting languages such as Novell Script For NetWare and Perl, C/C++ and Java.
- Reference count: UCX library manager automatically loads and unloads UCX NLMs, providing functionality similar to Dynamic Link Libraries. Handling libraries on an "as-needed" basis maximizes cache memory and achieves better server performance.
- Supports development of methods that can perform multiple tasks based on the data type and number of passed parameters.
- Provides runtime parameter type checking allowing applications to guard against abnormal termination due to type mismatch errors.
- Containment: Can share functionality and stack-up to create new components. One or more UCX components can be used to create a another UCX component.

1.1 UCX Component Structure

UCX library NLM consists of a main() function, an Unloader function, a set of declarations to declare properties, methods and events and a library procedure and methods.

1.1.1 main()

Executes upon loading of the UCX Library NLM. It initializes and then registers the UCX Component structure that describes the characteristics of the component.

1.1.2 Unloader

Unloads a UCX library NLM. A request to unload is triggered by either a client or the NSN engine. Unloader function ensures that a request to unload is not serviced if any other clients are accessing the library.

Syntax

```
int unloader()
```

Parameters

None.

Return values

Returns 1 if the library NLM is not unloaded.

Remarks

None.

See also

"main()" on page 9

1.1.3 UCX Component Structure

Every UCX component has an initialization structure describing its characteristics. When a library registers, the initialization structure is passed to UCX manager. One library NLM can contain the implementation for several UCX classes.

```
typedef struct tagUCXComponent{
   uint32    Signature;     // UCX Signature:Reserved
   uint32      MinVer;        // Minor version of UCX Component
   uint32      MajVer;        // Major version of UCX Component
   uint32     TimeStamp;    // timestamp for when
                            // component was
                            // last accessed
   uint32     LibNLMID;     // NLM ID of this Library
   uint32     LibScreenID;  // Screen ID for this library
   uint32     LibThreadID;  // Thread ID for the main
                            // thread
   uint32     LibThreadGroupID;// Thread Group ID for the
                               // main thread
   uint32     UnloadDelay;     // seconds since last use
                               // before component unloads
   sint32     NumClients;      // number of contexts using
                                 // this component
   sint32     NumClasses;      // number of classes that
                               // component registered
   UCXClass   *Class;          // Pointer to Class array
   puint8      ModuleName;     // Used for prefixing status
                               // messages.
   puint8      Message1;       // Message line one
   puint8      Message2;       // Message line two
   puint8      CopyRight;      // Copyright line
   uint8       Reserved[184];    // Reserved
} UCXComponent;
```

This structure should be included in the function "main()" on page 9, before UCXLibraryRegister() is called to register the UCX component. The macro UCXDEFINELIB is used to declare this structure.

1.1.4 UCX Data Types

UCX supports following data types.

```
UCX_DWORD_TYPE          unsigned long
UCX_DATE_TYPE             double. Integer portion represents the
                             days and fractional portion represents
                            the time of the day (fraction of day).
UCX_CURRENCY_TYPE       double
  UCX_SINGLE_TYPE        float
  UCX_LONG_TYPE          long
  UCX_USER_DEFINED_TYPE   User defined data type
  UCX_BYTE_TYPE          unsigned char
  UCX_WORD_TYPE          unsigned short
  UCX_ARRAY_TYPE         Array
  UCX_STRING_C_TYPE      C strings
  UCX_INTEGER_TYPE       sint16
  UCX_BOOLEAN_TYPE       unit32
  UCX_DOUBLE_TYPE        double
  UCX_STRING_TYPE        UCX string
  UCX_FILE_TYPE          Reserved
  UCX_OBJECT_TYPE        UCX Objects
  UCX_UNKNOWN_TYPE        Reserved
```

Following types are used to represent the property type or the method argument types.

```
  UCX_POINTER_TYPE       pointer type
  UCX_ANY_TYPE           variant type
  UCX_VARIANT_TYPE       Represents any data type
  UCX_OPTIONAL_TYPE      Represents the optional parameters
                         to methods
  UCX_VOID_TYPE           void
```

1.1.5 UCX Method Declaration

Each UCX component contains a methods table, which describes the methods that are implemented by one of the UCX classes. The following items are required to describe a method table.

- “UCX Parameter Structure” on page 11
- “Method Index Number” on page 12

UCX Parameter Structure

UCX parameter structure is used to declare input and output parameters of a method. The first item in the parameter structure must contain return type and name. Additional items in the structure represent the arguments passed to the method. The format of each entry is the UCX data type followed by the parameter name.NULL for both fields.

Each method in the UCX class will has its own parameter structure. The same parameter structure can be used for multiple libraries, if the return type and parameter types are identical for those libraries.

The following code snippet provides a sample for declaring the method Int KeyRead([int timeout]).

```
UCXParameter KeyReadParams[] = {
   {UCX_INTEGER_TYPE, "key_code"},
   {UCX_OPTIONAL_TYPE, "timeout"},
   {NULL, NULL}
};
```

Method Index Number

Each method is assigned an index in the class method list. Index positions begin at 0.

It is always recommended to define the enum labels using an uppercase version of the actual method name. For example, the method "READ" for the KEY object would have the enum name "KEY_READ". This convention makes it easy to identify the method referenced within in the code.

The following code snippet gives a sample library function list.

```
enum
   {
      KEY_READ,
      KEY_READY,
      KEY_WRITE
   };
```

Declaring the Methods Table

The following code snippet declares the methods READ, READ and WRITE.

Note that the methods table is also terminated with NULL elements for the last row like the parameter list. Ensure that the methods in the table are capitalized and sorted to prevent the failure of UCX function lookup API.

```
   UCXMethod MyMethods[] = {
      {"READ", KEY_READ, KeyReadParams},
      {"READY", KEY_READY, KeyReadyParams},
      {"WRITE", KEY_WRITE, KeyWriteParams},
      {NULL, NULL, NULL}
   };
```

NOTE: The methods in the methods table must be capitalized and sorted. Else, the UCX function lookup API fails.

1.1.6 UCXProperty Declaration

UCX Property Table

Each UCX component contains a property table, which describes the properties that are implemented by one of the UCX classes.

The following section describes each of the elements in more detail.

UCX Property Structure

The UCXProperty structure describes each property. The property structure includes

- UCX data type: The data type of the property.
- Property name: The name of the property.

- Property scope: The scope of the property.

  The two scopes currently supported by UCX are UCX_PUBLIC and UCX_PRIVATE. Properties having scope UCX_PRIVATE, can be read or written, only by the component. These properties are used to declare hidden properties of the components. Properties having the scope UCX_PUBLIC can be read and written by the user also.

Like most other UCX tables, the last row of the UCX property structure contains NULL for all fields.

The following code snippet gives a sample UCX property structure.

```
UCXProperty MyPropertyList[] = {
   {UCX_INTEGER_TYPE, "LAST", UCX_PUBLIC},
   {0, NULL, 0}
};
```

This sample describes a single property with property name "LAST", which has property scope of public data type UCX_INTEGER_TYPE.

1.1.7 UCX Class Declaration

UCX Class Table

Each UCX component contains a class table, which describes the classes that are implemented within the library. The class table describes all the elements required for a specific class. Some of the elements that can be included in the UCX class table are

- Methods
- Properties
- Events
- Library procedure
- Class name

The following code snippet provides a sample UCX class table.

In the sample the number of objects, methods, events and properties are all specified as zero. They will be initialized by UCX during registration of the class.

The class structure includes a pointer referring to the UCX component structure. The Methods, Events and Properties contain pointers to the respective tables. A pointer to the message handler function for the class is also provided. Multiple classes can be defined by a single component, by providing multiple entries in the class table. The table is terminated, by a providing a NULL pointer for the Classname member.

```
UCXClass MyClassList[] =  {
{"UCX:KEY", 0, 0, 0, 0, &MyLib, MyMethods, NULL,
 MyPropertyList, UCXLibProc,NULL},
 {NULL}
};
```

UCX Class Structure

The UCXClass structure describes each class supported by a specific component.

The following code snippet provides a sample UCX class structure.

```
typedef struct{
      puint8       ClassName;      // class name
      sint32       NumObjects;     // number of objects
                                   // currently in scope
      uint32       NumMethods;     // total number
                                   // of methods
      uint32       NumEvents;      // total number of events
      uint32       NumProperties;  // total number of
                                   // properties
      UCXComponent *Component;     // Pointer to the
                                   // component
                                   // structure that
                                   // implements the class
      UCXMethod     *Methods;      // methods
      UCXEvent      *Events;       // events
      UCXProperty   *Properties;   // properties
      UCXC_API      (*UCXLibProc)(void *, void *, puint8,
                    uint32, struct __ClassLnk *, void *);
      puint8            pDefaultItem;    // Name of the default
                                   // property or method.
   } UCXClass;
```

1.2 UCX Library Procedure

UCX library procedure (UCXLibProc)or message dispatcher handles all the events generated for a UCX component. A pointer to the UCXLibProc is included in the class definition structure. If UCXLibProc does not handle a generated event, it is passed to the UCX function UCXDispatchEvent for default handling. Usage of the name UCXLibProc is optional.

The prototype for the library handler is:

```
Void* UCXLibProc(void *CP,
  void *Params,
  Char *FN,
  uint32 Event,
  UCXClassLink *ClassLink,
  void *Object);
```

The UCX Library Procedure is called to handle the various events generated by UCX library manager. The meaning of the parameters to the library procedure varies based on the type of event. The meaning of each parameter in the library procedure is discussed in the successive sections of this documentation.

The standard UCX events passed to UCXLibProc() are

- UCX_EVENT_LIB_BIND
- “UCX_EVENT_LIB_UNBIND” on page 16
- “UCX_EVENT_CLASS_CREATE” on page 16
- “UCX_EVENT_CLASS_DESTROY” on page 17
- “UCX_EVENT_BIND” on page 17

The miscellaneous UCX events passed to UCXLibProc() are

1.2.1 UCX_EVENT_LIB_BIND

Is called only once when the library is first loaded in the memory. After the library has been loaded successfully, UCX will send this event to the library to make sure that the library can initialize. This event is called on the library thread ID.

For example a library that sends and receives faxes may want to check if the fax services or the proper hardware is installed before continuing. This event must return UCX True if successful or UCX False if it failed.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

The function name which caused the component to load.

Event

UCX_EVENT_LIB_BIND.

ClassLink

A pointer to the Global UCXClassLink structure.

Object

Not applicable.

1.2.2 UCX_EVENT_LIB_UNBIND

Is dispatched only once when the library is unloading. If the library allocated memory or opened any semaphores in UCX_EVENT_LIB_BIND routine, this would be the place to de-allocate those resources. This event is called on the library thread ID.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

Not applicable.

Event

UCX_EVENT_LIB_UNBIND.

ClassLink

A pointer to the Global UCXClassLink structure.

Object

Not applicable.

1.2.3 UCX_EVENT_CLASS_CREATE

Registered class proxy NLMs are notified when the UCX client is attempting to instantiate an object based on a class prefix that they have registered. Whenever NLMS are notified, this event is sent to the proxy so that it may dynamically register the class that the client is requesting.

The return value from this event should contain the named class, else the client will receive a failure indication for the UCXInstantiateObject() API call.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

The name of the class the client is attempting to create.

Event

UCX_EVENT_CLASS_CREATE.

ClassLink

Not applicable.

Object

Not applicable.

1.2.4 UCX_EVENT_CLASS_DESTROY

Is dispatched once a class has been destroyed, and removed from the internal UCX class lists. A message handler can capture this event, and perform any additional cleanup if required.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

Not applicable.

Event

UCX_EVENT_CLASS_CREATE.

ClassLink

The Global ClassLink referring to the class getting destroyed.

Object

Not applicable.

1.2.5 UCX_EVENT_BIND

Processes a client of the UCX library. A library receives this event when an unregistered client first makes a call to the library. Normally this event should be passed to UCXDispatchEvent for default handling. This event is called on the calling client thread ID.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

The function name that caused this event to get dispatched.

Event

UCX_EVENT_BIND.

ClassLink

A pointer to the Global UCXClassLink structure.

Object

Not applicable.

1.2.6 UCX_EVENT_UNBIND

De-registers a client from an UCX library. A library receives the event when a client de-registers with the library or the client process terminates. Normally this event should be passed to UCXDispatchEvent for default handling. This event is called on the calling client thread ID.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

Not applicable.

Event

UCX_EVENT_UNBIND.

ClassLink

A pointer to the Global UCXClassLink structure.

Object

Not applicable.

1.2.7 UCX_EVENT_CLIENT_CREATE

Allocates memory for a registered client. A library receives the event when a registered client first makes a call to the library. Pass the event to UCXDispatchEvent for default handling if memory initialization is unnecessary.

This event is dispatched once for every client. A pointer called UCXLibLink->ClientData can be used to store any type of information for each individual client. An integer called UCXLibLink->ClientTag is also available.

If memory is to be allocated when a client starts using a library, this event must be captured. Every resource that is allocated in this event must be freed in the UCX_EVENT_CLIENT_DESTROY event. This event is called on the calling client thread ID.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

The name of the class that is being referenced.

Event

UCX_EVENT_CLIENT_CREATE.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

Not applicable.

1.2.8 UCX_EVENT_CLIENT_DESTROY

De-allocates memory associated with a registered client. A library receives this event when a registered client requests to de-register or the client process terminates. Pass the event to UCXDispatchEvent for default handling if memory de-allocation is unnecessary. This event is called on the calling client thread ID. All resources allocated in UCX_EVENT_CLIENT_CREATE must be freed during this event. The pointer UCXLibLink->ClientData is normally used to point to per-client data.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

Not applicable.

Event

UCX_EVENT_CLIENT_DESTROY.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

Not applicable.

1.2.9 UCX_EVENT_OBJECT_CREATE

Is raised when a new object is created based on the class that the message handler is servicing requests for. When this event occurs, the object has been created, and all properties added with zero values. The component can capture this event to provide additional initialization for the object properties. Library procedure returns FALSE when object creation fails.

Parameters

CP

The UCX Context Pointer.

Params

Normally a NULL pointer, but is non-NULL if the caller of UCXInstantiateObject() passes in a non-null 4th parameter.

FN

The name of the class this object belongs to.

Event

UCX_EVENT_OBJECT_CREATE.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that was just created.

1.2.10 UCX_EVENT_OBJECT_DESTROY

Is raised when an object is going out of scope. This event occurs before the object is destroyed.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

Not applicable.

Event

UCX_EVENT_OBJECT_DESTROY.

ClassLink

Not applicable.

Object

The object that is getting destroyed.

1.2.11 UCX_EVENT_PROPERTY_GET

Is generated when a property value is being read. The component captures this event overrides the default behavior of simply returning the current value.

Parameters

CP

The UCX Context Pointer.

Params

Not applicable.

FN

The name of the property being read.

Event

UCX_EVENT_CLASS_CREATE.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that is being manipulated.

1.2.12 UCX_EVENT_PROPERTY_SET

Is generated when a property value is being modified. The component captures this event and overrides the default behavior.

Parameters

CP

The UCX Context Pointer.

Params

The new value being set into the property.

FN

The name of the property being written.

Event

UCX_EVENT_PROPERTY_SET.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that is being manipulated.

1.2.13 UCX_EVENT_PARAM_GET

Gets the parameter list for a UCX library method. This event should be passed to UCXDispatchEvent for default handling. This event is called on the calling client thread ID.

Parameters

CP

The UCX Context Pointer.

Params

The parameters that are being passed to the method.

FN

The name of the method being invoked.

Event

UCX_EVENT_PARAM_GET.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that is being manipulated.

1.2.14 UCX_EVENT_EXEC

Requests execution of a UCX class method. This event is called on the calling client thread ID.

Parameters

CP

The UCX Context Pointer.

Params

The parameters that are being passed to the method.

FN

The name of the method being invoked.

Event

UCX_EVENT_EXEC.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that is being manipulated.

1.2.15 UCX_EVENT_CONSTANT

Gets the list of constants required by the component. Scripting languages take care of making these constants as script constants.

This event is called when an object is created. Library procedure returns the constant list, if any of its properties or methods require constants.

Parameters

FN

The name of the class this object belongs to.

Event

UCX_EVENT_CONSTANT.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that is being manipulated.

Example

Declaration of UCX constants is similar to the declaration of UCX Methods. It has following structure.

```
typedef struct{
  uint32 Type; //Data type of the constant
  uint8  *Name;  //Name of the constant
  uint8  *Value; //Value of the constant represented as a string
} UCXConstant;
```

UCX Manager takes care of converting the value of the constant to the required UCX data type.

1.2.16 UCX_EVENT_DEFAULT_ITEM

Is sent by the clients to retrieve the name of the default property or method. The default item declared in the class structure is overridden by handling this event.

Parameters

CP

The UCX Context Pointer.

Params

The parameters that are being passed to the default method or property.

FN

The name of the class this object belongs to.

Event

UCX_EVENT_DEFAULT_ITEM.

ClassLink

A pointer to the Context-level UCXClassLink structure.

Object

The object that is being manipulated.

1.2.17 UCX_EVENT_CONTEXT_DESTROY

Notifies all the global objects that the context is getting destroyed or client application is terminating.

Parameters

None.

1.2.18 Miscellaneous Events

Services events generated by other components.

1.3 Event Default Handling

The following logic illustrates the default handling for each of the UCX Event types. UCX manager exports this default handler function.

```
   switch( Event ) {
   case UCX_EVENT_PARAM_GET:
      if( (i = UCXMethodIndex(CP, ClassLink->Class, FN)) !=
1 ) {
      return UCXCheckParam(CP, ClassLink->Class, i);
    }
      return NULL;

   case UCX_EVENT_PROPERTY_GET:
      return __ReadPropertyValue(CP,FN,Object);

   case UCX_EVENT_PROPERTY_SET:
      return __WritePropertyValue(CP,FN,Object,VoidPtr);

   case UCX_EVENT_BIND:
      if( UCXRegisterClient(CP, ClassLink->Class) ) {
         return UCXCreateBoolean(CP, TRUE, FN);
      }
      return UCXCreateBoolean(CP, FALSE, FN);

   case UCX_EVENT_UNBIND:
      i = UCXDeregisterClient(CP, ClassLink->Class);
      return(UCXCreateBoolean(CP, !i, FN));

   case UCX_EVENT_CLIENT_CREATE:
      return(UCXCreateBoolean(CP, TRUE, FN));

   case UCX_EVENT_CLIENT_DESTROY:
      return(NULL);

   case UCX_EVENT_OBJECT_CREATE:
      return(UCXCreateBoolean(CP, TRUE, FN));

   case UCX_EVENT_OBJECT_DESTROY:
      return(NULL);

   case UCX_EVENT_CLASS_CREATE:
      return(UCXCreateBoolean(CP, FALSE, FN));

   case UCX_EVENT_CLASS_DESTROY:
      return(NULL);

   case UCX_EVENT_LIB_BIND:
      return(UCXCreateBoolean(CP, TRUE, FN));

   case UCX_EVENT_LIB_UNBIND:
      return(UCXCreateBoolean(CP, TRUE, FN));

   case UCX_EVENT_EXEC:
      return(NULL);

   case UCX_EVENT_SYNC_HANDLER:
         return NULL;

   case UCX_EVENT_ASYNC_HANDLER:
         return NULL;

   case UCX_EVENT_CONSTANT:
      return(NULL);
```

```
case UCX_EVENT_CONTEXT_DESTROY:
   return(NULL);
}
```

Tasks

2

This section provides details about

2.1 Developing UCX Components

Proceed with the following steps to develop an UCX component.

1. Identify the requirements and the features of the component.
2. Design the component. Identify the properties, methods and events.
3. Declare the method list, property list, constants list and class structure.
4. Declare the library procedure.
5. Handle necessary UCX events to perform various actions for invoking different methods or properties.
6. Compile the code using any C/C++ compiler and link with the required libraries.

2.2 Distributing UCX Components

UCX component NLMs can be copied to any directory on the NetWare server. The convention is to keep the UCX components under `SYS:UCS\UCX` directory. UCX components are registered with the UCX manager through a configuration file `UCX.INI` located in `SYS:UCS\UCX` directory. Include the name and path of the UCX component in this configuration file. For example, the entry to register UCX:KEY component is

```
UCX:KEY=SYS:UCS\UCX\KEY.NLM
```

2.3 Implementing Collection

Collection represents a group of objects. For example, Files can be a collection of file objects. Novell Script For NetWare language has a For - Each syntax for easy navigation through collections.

The collection class has to implement following methods for easy navigation using For - Each syntax.

2.3.1 Reset method

Resets the collection.

Syntax

```
void Reset()
```

Parameters

None.

Return Values

None.

Remarks

None.

2.3.2 Next method

Retrieves the next object from the collection.

Syntax

```
object Next()
```

Parameters

None.

Return values

Returns the next object from the collection.

Remarks

None.

2.3.3 HasMoreElements method

Determines whether or not any more objects exist in the collection.

Syntax

```
boolean HasMoreElements ()
```

Parameters

None.

Return values

Boolean. Returns TRUE if collection has more elements else returns FALSE.

Remarks

None.

UCX SDK Functions

3

UCX API includes a set of functions that are classified as follows

- Section 3.1, “Component Functions,” on page 29 - Used to create UCX components.
- Section 3.2, “Client Functions,” on page 61 - Used by the application, which makes use of UCX components.
- Section 3.3, “Miscellaneous Functions,” on page 87 - Used for miscellaneous functions like error handling. Components as well as clients can use these functions.

3.1 Component Functions

Library builder API or Component API are used to create UCX components.

The following data types are used in the function definitions of the component component API.

```
typedef signed   char sint8;
typedef unsigned char uint8;
typedef signed   short sint16;
typedef unsigned short uint16;
typedef signed   long sint32;
typedef unsigned long uint32;
typedef float    ssingle;
typedef double   sdouble;
typedef unsigned char CHAR;

typedef signed   char *psint8; // pointers matching
                                  above types
typedef unsigned char *puint8;
typedef signed   short *psint16;
typedef unsigned short *puint16;
typedef signed   long *psint32;
typedef unsigned long *puint32;
typedef float          *pssingle;
typedef double         *psdouble;
typedef unsigned char   *CHAR_PTR;
```

The component API can be classified as

- “Component Builder Functions” on page 29 - used to develop UCX components.
- “Parameter Functions” on page 33 - used for extracting parameters from the parameter list.
- “Property Functions” on page 44 - used to set and get the properties of the UCX components.

3.1.1 Component Builder Functions

This section provides details about the various component builder functions. The component builder functions are:

UCXRegisterComponent
UCXDeregisterComponent
UCXGetClientClassLink

UCXMethodIndex
UCXDispatchEvent
UCXCreateParam
UCXDestroyParam
UCXGetArrayParam
UCXGetBooleanParam
UCXGetByteParam
UCXGetParamCount
UCXGetCurrencyParam
UCXGetDateParam
UCXGetDoubleParam
UCXGetDWordParam
UCXGetIntegerParam
UCXGetLongParam
UCXGetSingleParam
UCXGetRVPtrParam
UCXGetStringParam
UCXGetParamType
UCXGetWordParam

UCXRegisterComponent

Registers a UCX component.

Syntax

```
sint32 UCXRegisterComponent(void *CP, UCXComponent *Component)
```

Parameters

CP

(IN) Context ID of caller

Component

(IN) The UCX Library structure for the library that is registering with UCX.

Return Value

Returns 1 if the UCX Library registration process succeeds else returns -1.

Remarks

UCXRegisterComponent is part of the internal framework of every UCX library. Prior to making an UCX library function call, the library must be loaded into memory and initialized. UCXRegisterComponent initializes the UCX library.

See Also

"UCXDeregisterComponent" on page 31

UCXDeregisterComponent

De-registers the UCX component.

Syntax

```
sint32 UCXDeregisterComponent(void *CP, UCXComponent *Component)
```

Parameters

CP

(IN) Context ID of caller

Component

(IN/OUT) UCX class structure pointer

Return Value

Returns TRUE if component de-registration suceeds.

Remarks

This function returns TRUE and starts the Component unload if the component has no clients.

See also

"UCXRegisterComponent" on page 30

UCXGetClientClassLink

Gets a pointer to the local class structure of a registered UCX class.

Syntax

```
UCXClassLink *UCXGetClientClassLink (void *CP, uint32 LockFirst, char
*ClassName)
```

Parameters

CP

(IN) Context ID of caller

LockFlag

(IN) TRUE to lock global list semaphore first

ClassName

(IN) The name of the UCX Class to retrieve the client link for.

Return Value

Returns a pointer to the UCXClassLink structure for the retrieved class name else returns NULL.

Remarks

Locate the client class link for 'ClassName'. The class link returned by this API is from the context specified. Each UCX context maintains a list of all classes currently being used by the context. Per context client data is stored within the ClientData and ClientTag members of the link. This API is intended for use by internal UCX functions only.

UCXMethodIndex

Get the index of a UCX method from the class method table.

Syntax

```
sint32 UCXMethodIndex(void *CP, UCXClass *Class, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

Class

(IN) A pointer to the UCX Library structure whose function table is to be searched

FN

(IN) UCX method name that we want to locate

Return Value

Returns the index of the function specified by FN else returns -1.

Remarks

UCX components maintain a methods table, which contains the string name, numeric ID and UCX parameter list of each UCX method supported by the components. UCXMethodIndex returns the array index assigned to the UCX method named in FN.

UCXDispatchEvent

Is the default event handler for UCX libraries

Syntax

```
void *UCXDispatchEvent (void *CP, void *Params, CHAR_PTR FN, uint32 Event,
UCXClassLink *ClassLink, void *Object)
```

Parameters

CP

(IN) Context ID of caller

Params

(IN) The input parameters which were passed to the UCX class's registered event handler.

FN

(IN) UCX API Name that is executing.

Event

(IN) The UCX Event type that was passed to the registered event handler.

ClassLink

(IN) The Library link structure pointer that was passed to the registered event handler

Object

(IN)The pointer to UCX object

Return Value

Returns an UCX RV (RETURN VALUE) pointer.

Remarks

UCXDispatchEventUCXDispatchEvent handles events which are not processed by an UCX class's event handler. The events are passed back to the UCX engine for default handling. In many cases, events are raised which a given UCX class does not need to capture, and use of this API eliminates the need for redundant event handling code in each UCX class. In addition, as new event types are added to UCX, use of this API enables upward compatibility with newer implementations of the UCX Library manager.

3.1.2 Parameter Functions

This section provides details about the various parameter functions. The parameter functions are:

UCXCreateParam
UCXDestroyParam
UCXGetArrayParam
UCXGetBooleanParam
UCXGetByteParam
UCXGetParamCount
UCXGetCurrencyParam
UCXGetDateParam
UCXGetDoubleParam
UCXGetDWordParam
UCXGetIntegerParam
UCXGetLongParam
UCXGetSingleParam
UCXGetRVPtrParam
UCXGetStringParam
UCXGetStringParamLength
UCXGetParamType
UCXGetWordParam

UCXCreateParam

Creates a UCX parameter list, suitable for passing to UCXCall().

Syntax

```
void *UCXCreateParam(void *CP, char *FN,....)
```

Parameters

CP

(IN) Context ID of caller

FN

(IN) UCX API Name that you want to execute

(variable argument list)

(IN) UCX Data Type/Argument Value pairs

Return Value

Returns a pointer to an UCX Parameter structure, suitable for passing to UCXCall(). Returns NULL if an error is encountered, or the system is out of memory.

Remarks

Convert a list of C/C++ types to an UCX parameter list. Note that character strings must be converted to UCX strings, before passing them to UCXCreateParam(). This call is intended for developers who need to construct UCX parameter lists by hand, instead of using the automatic conversion provided by UCXEZCall.

NOTE: UCXCreateParam must be terminated with two NULL Parameters.

See also

"UCXCall" on page 70
"UCXDestroyParam" on page 34

UCXDestroyParam

Frees a UCX parameter list.

Syntax

```
void UCXDestroyParam (void *CP, void *ParamList)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) Pointer to a UCX parameter list, created with UCXCreateParam()

Return Value

None.

Remarks

Release the memory associated with the specified UCX parameter list.

See also

"UCXCall" on page 70
"UCXCreateParam" on page 34

UCXGetArrayParam

Retrieves a pointer to an array value from a UCX parameter list.

Syntax

```
void *UCXGetArrayParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns a pointer corresponding to the specific array type or returns NULL if no array or parameter is present corresponding to the specific array type.

Remarks

Obtain a pointer to the array value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43
"UCXCreateArray" on page 62

UCXGetBooleanParam

Retrieves a Boolean value from a UCX parameter list.

Syntax

```
uint32 UCXGetBooleanParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns a non-zero value if TRUE is retrieved else returns ZERO.

Remarks

Gets the Boolean value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetByteParam

Retrieves a Byte value from a UCX parameter list.

Syntax

```
uint32 UCXGetByteParam(void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved byte value else returns NULL.

Remarks

Gets the Byte value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34

“UCXGetParamType” on page 43

UCXGetParamCount

Get a count of the number PARAMETERS in a UCX parameter list.

Syntax

```
uint32 UCXGetParamCount (void *CP, void *ParamList)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Return Value

An unsigned integer specifying how many PARAMETERS are stored in the ParamList structure.

Remarks

UCX parameter lists are used to pass arguments to UCX library functions. This API is used to determine how many PARAMETERS have been passed, and is used by functions which allow optional arguments to be passed.

See also

“UCXGetParamType” on page 43

UCXGetCurrencyParam

Retrieves a currency value from a UCX parameter list

Syntax

```
uint32 UCXGetCurrencyParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved currency value else returns NULL.

Remarks

Gets the currency value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetDateParam

Retrieves a date value from a UCX parameter list.

Syntax

```
uint32 UCXGetDateParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved date value else returns NULL.

Remarks

Gets the date value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetDoubleParam

Retrieves a double value from a UCX parameter list

Syntax

```
uint32 UCXGetDoubleParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved double value else returns NULL.

Remarks

Gets the double value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetDWordParam

Retrieves a Dword value from a UCX parameter list

Syntax

```
uint32 UCXGetDWordParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved DWord value else returns NULL.

Remarks

Gets the DWord value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetIntegerParam

Retrieves a integer value from a UCX parameter list

Syntax

```
uint32 UCXGetIntegerParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved integer value else returns NULL.

Remarks

Gets the integer value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetLongParam

Retrieves a long value from a UCX parameter list

Syntax

```
uint32 UCXGetLongParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved long value else returns ZERO.

Remarks

Gets the long value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetSingleParam

Retrieves a single value from a UCX parameter list.

Syntax

```
uint32 UCXGetSingleParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved single value else returns ZERO.

Remarks

Gets the single value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetRVPtrParam

Gets the RV type from a UCX parameter list.

Syntax

```
void *UCXGetRVPtrParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns a pointer to a UCX RV type stored at the specified position in the parameter list.

Remarks

An RV pointer addresses a single type in a UCX parameter list. UCXGetRVPtrParam returns the RV pointer referenced by the provided index. Normally, programmers will extract the data element directly, using other NXMParam... functions. This API is provided for completeness, however, allowing the developer to obtain the RV structure itself, should the need arise.

UCXGetStringParam

Retrieves a single value from a UCX parameter list

Syntax

```
uint32 UCXGetStringParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved string value else returns NULL.

Remarks

Gets the string value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetStringParamLength

Retrieves a string length value from a UCX parameter list

Syntax

```
uint32 UCXGetStringParamLength (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved string length value else returns ZERO.

Remarks

Gets the string length value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

UCXGetParamType

Gets a UCX parameter list type.

Syntax

```
uint32 UCXGetParamType (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

A constant representing the data type of the parameter stored at the specified position in the parameter list. Possible types are

- UCX_STRING_C_TYPE
- UCX_INTEGER_TYPE
- UCX_LONG_TYPE
- UCX_BOOLEAN_TYPE

- UCX_REAL_TYPE
- UCX_STRING_TYPE
- UCX_VOID_TYPE
- UCX_UNKNOWN_TYPE

Remarks

Returns a constant representing the parameter data type in a UCX parameter list. See (include section in appendix) and provide link *****

See also

"UCXGetParamCount" on page 37

UCXGetWordParam

Retrieves a word from a UCX parameter list

Syntax

```
uint32 UCXGetWordParam (void *CP, void *ParamList, uint32 Index)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list containing all PARAMETERS passed to the API

Index

(IN) The index position in the parameter list which contains the parameter

Return Value

Returns the retrieved word else returns ZERO.

Remarks

Gets the word value at index position Index from the UCX parameter list specified by ParamList.

See also

"UCXCreateParam" on page 34
"UCXGetParamType" on page 43

3.1.3 Property Functions

This section provides details about the various property functions. The property functions are:

UCXGetProperty
UCXSetProperty
UCXGetIntegerProperty

UCXSetIntegerProperty
UCXGetByteProperty
UCXSetByteProperty
UCXGetLongProperty
UCXSetLongProperty
UCXGetWordProperty
UCXSetWordProperty
UCXGetDWordProperty
UCXSetDWordProperty
UCXGetDoubleProperty
UCXSetDoubleProperty
UCXGetSingleProperty
UCXSetSingleProperty
UCXGetDateProperty
UCXSetDateProperty
UCXGetCurrencyProperty
UCXSetCurrencyProperty
UCXGetStringProperty
UCXSetStringProperty
UCXGetPointerProperty
UCXSetPointerProperty
UCXGetPropertyType
UCXGetPropertyLV

UCXGetProperty

Gets a copy of a property value from an object.

Syntax

```
void *UCXGetProperty(void *CP, void *obj, char *name);
```

Parameters

CP

(IN) Context ID of caller

name

(IN) Name of the property

obj

(IN) Object structure

Return Value

Returns the pointer corresponding to the specific property else returns NULL.

Remarks

A copy of the property value named ’name’ in object ’obj’ is returned to the caller. The memory associated with this value must be freed with UCXFreeRV(), when it is no longer needed.

UCXSetProperty

Sets a new value to a property of an object.

Syntax

```
int UCXSetProperty(void *CP, void *obj, char *name, void *newValue);
```

Parameters

CP

(IN) Context ID of caller

name

(IN) Name of the property

obj

(IN) Object structure

newValue

(IN) New value to assign to the property. Construct this parameter using UCXCreateParam(), as it must follow the standard parameter passing convention for UCX

Return Value

Returns TRUE if the property is set else returns FALSE.

Remarks

The return value is based on whether the property ’name’ was found in the object ’obj’, and the value ’newValue’ was set into the object. ’newValue’ replaces the current property value, and thus should not be freed upon return.

UCXGetIntegerProperty

Retrieves the value of the integer property.

Syntax

```
int UCXGetIntegerProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific integer value is retrieved else returns -1.

Remarks

Fills in num with integer value of the property with the specified name.

See also

"UCXSetIntegerProperty" on page 47

UCXSetIntegerProperty

Sets the value of the integer property.

Syntax

```
int UCXSetIntegerProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific integer value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetIntegerProperty" on page 46

UCXGetByteProperty

Retrieves the value of the byte property.

Syntax

```
int UCXGetByteProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific byte value is retrieved else returns -1.

Remarks

Fills in num with byte value of the property with the specified name.

See also

"UCXSetByteProperty" on page 48

UCXSetByteProperty

Sets the value of the byte property.

Syntax

```
int UCXSetByteProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific byte value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetByteProperty" on page 48

UCXGetLongProperty

Retrieves the value of the long property.

Syntax

```
int UCXGetLongProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the long value of the property with the specified name.

See also

"UCXSetLongProperty" on page 49

UCXSetLongProperty

Sets the value of the long property.

Syntax

```
int UCXSetLongProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific long value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetLongProperty" on page 49

UCXGetWordProperty

Retrieves the value of the word property.

Syntax

```
int UCXGetWordProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the word value of the property with the specified name.

See also

"UCXSetWordProperty" on page 51

UCXSetWordProperty

Sets the value of the word property.

Syntax

```
int UCXSetWordProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific word value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetWordProperty" on page 50

UCXGetDWordProperty

Retrieves the value of the Dword property.

Syntax

```
int UCXGetDWordProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the Dword value of the property with the specified name.

See also

"UCXSetDWordProperty" on page 52

UCXSetDWordProperty

Sets the value of the Dword property.

Syntax

```
int UCXSetDWordProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific Dword value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetDWordProperty" on page 51

UCXGetDoubleProperty

Retrieves the value of the double property.

Syntax

```
int UCXGetDoubleProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the double value of the property with the specified name.

See also

"UCXSetDoubleProperty" on page 53

UCXSetDoubleProperty

Sets the value of the double property.

Syntax

```
int UCXSetDoubleProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific integer value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetDoubleProperty" on page 53

UCXGetSingleProperty

Retrieves the value of the single property.

Syntax

```
int UCXGetSingleProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific single value is retrieved else returns -1.

Remarks

Fills in num with the single value of the property with the specified name.

See also

"UCXSetSingleProperty" on page 55

UCXSetSingleProperty

Sets the value of the single property.

Syntax

```
int UCXSetSingleProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific single value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetSingleProperty" on page 54

UCXGetDateProperty

Retrieves the value of the date property.

Syntax

```
int UCXGetDateProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific date value is retrieved else returns -1.

Remarks

Fills in num with the date value of the property with the specified name.

See also

"UCXSetDateProperty" on page 56

UCXSetDateProperty

Sets the value of the date property.

Syntax

```
int UCXSetDateProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific date value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetDateProperty" on page 55

UCXGetCurrencyProperty

Retrieves the value of the currency property.

Syntax

```
int UCXGetcurrencyProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the currency value of the property with the specified name.

See also

"UCXSetCurrencyProperty" on page 57

UCXSetCurrencyProperty

Sets the value of the currency property.

Syntax

```
int UCXSetCurrencyProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific currency value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetCurrencyProperty" on page 56

UCXGetStringProperty

Retrieves the value of the string property.

Syntax

```
int UCXGetStringProperty(void *CP, void *obj, char *name, int *num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(OUT) Integer value of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the string value of the property with the specified name.

See also

"UCXSetStringProperty" on page 58

UCXSetStringProperty

Sets the value of the string property.

Syntax

```
int UCXSetStringProperty(void *CP, void *obj, char *name, int num)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

num

(IN) Integer value of the property

Return Value

Returns 1 if the specific string value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetStringProperty" on page 58

UCXGetPointerProperty

Retrieves the value of the pointer property.

Syntax

```
int UCXGetPointerProperty(void *CP, void *obj, char *name)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

Return Value

Returns 1 if the specific long value is retrieved else returns -1.

Remarks

Fills in num with the pointer value of the property with the specified name.

See also

"UCXSetPointerProperty" on page 60

UCXSetPointerProperty

Sets the value of the pointer property.

Syntax

```
int UCXSetPointerProperty(void *CP, void *obj, char *name, void *p)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

p

(IN) Pointer to which the property is set

Return Value

Returns 1 if the specific integer value is set else returns -1.

Remarks

Fills in value of the property with the specified name with value of num.

See also

"UCXGetIntegerProperty" on page 46

UCXGetPropertyType

Retrieves the data type of the specified property.

Syntax

```
int UCXGetPropertyType(void *CP, void *obj, char *name)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

Return Value

Returns 1 if the data type of the specified property is retrieved else returns -1.

Remarks

None.

UCXGetPropertyLV

Retrieves the value of the long property.

Syntax

```
int UCXGetPropertyLV(void *CP, void *obj, char *name)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Object identifier

name

(IN) Name of the property

Return Value

****Returns 1 if the specific long value is retrieved else returns -1.

Remarks

****Fills in num with the long value of the property with the specified name.

See also

"UCXSetIntegerProperty" on page 47****

3.2 Client Functions

Client functions are utilized by applications that make use of UCX components.The client functions are:

UCXCreateArray
UCXCreateBoolean
UCXCreateByte
UCXCreateCurrency
UCXCreateDate
UCXCreateDWord

UCXCreateInteger
UCXCreateLong
UCXCreateDouble
UCXCreateSingle
UCXCreateString
UCXCreateWord
UCXCreatePointer
UCXCall
UCXCreateContext
UCXDestroyContext
UCXEZCall
UCXGetArrayFromRV
UCXGetBooleanFromRV
UCXGetByteFromRV
UCXGetCurrencyFromRV
UCXGetDateFromRV
UCXGetDoubleFromRV
UCXGetDWordFromRV
UCXFreeRV
UCXGetIntegerFromRV
UCXGetLongFromRV
UCXGetSingleFromRV
UCXGetStringFromRV
UCXGetStringElementFromRV
UCXGetStringLengthFromRV
UCXGetTypeFromRV
UCXGetUserTypeFromRV
UCXGetWordFromRV
UCXCopyContext
UCXDeInitializeContext
UCXLoadClass
UCXCreateObject
UCXInstantiateObject
UCXLocateObject
UCXInstantiateGlobalObject
UCXGetDefaultItem

3.2.1 UCXCreateArray

Converts an array to a UCX RV (return value) type.

Syntax

```
void *UCXCreateArray (void *CP, void *tv, void *Typeheader, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

tv

(IN)*

Typeheader

(IN)*

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX array return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.2 UCXCreateBoolean

Converts a boolean value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateBooean (void *CP, uint32 num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The boolean value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX boolean return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.3 UCXCreateByte

Converts a byte value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateByte (void *CP, uint8 num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The byte value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX byte return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.4 UCXCreateCurrency

Converts a currency value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateCurrency (void *CP, sdouble num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The currency value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX currency return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.5 UCXCreateDate

Converts a date value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateDate (void *CP, sdouble num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The date value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX date return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.6 UCXCreateDWord

Converts a Dword value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateDword (void *CP, uint32 num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The Dword value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX Dword return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.7 UCXCreateInteger

Converts an integer value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateInteger (void *CP, sint16 num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The integer that you want to return.

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX integer return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.8 UCXCreateLong

Converts a long value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateLong (void *CP, sint32 num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The long value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX long return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.9 UCXCreateDouble

Converts a double value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateDouble (void *CP, sdouble num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The double value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX double return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.10 UCXCreateSingle

Converts a single value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateSingle (void *CP, ssingle num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The single value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX single return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.11 UCXCreateString

Converts a string value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateString (void *CP, puint8 str, uint32 len, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

str

(IN) The string value to be returned

len

(IN) The length of the string

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX string return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.12 UCXCreateWord

Converts a word value to a UCX RV (return value) type.

Syntax

```
void *UCXCreateword (void *CP, uint16 num, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

num

(IN) The word value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX word return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.13 UCXCreatePointer

Converts a pointer value to a UCX RV (return value) type.

Syntax

```
void *UCXCreatePointer (void *CP, void *ptr, CHAR_PTR FN)
```

Parameters

CP

(IN) Context ID of caller

ptr

The pointer value to be returned

FN

(IN) UCX API Name that is creating the return value

Return Value

Returns the pointer corresponding to the UCX RV type else returns NULL.

Remarks

This API creates a UCX pointer return value, which is used to return data to a UCX client. UCX component functions return values to their clients by calling UCXCreate...() functions to encapsulate the data in UCX format.

3.2.14 UCXCall

Gives C/C++ programs access to UCX methods.

Syntax

```
void *UCXCall (void *CP, void *ParamList, void *object, CHAR_PTR FN, uint32
normal)
```

Parameters

CP

(IN) Context ID of caller

ParamList

(IN) A UCX parameter list constructed with UCXCreateParam()

object

(IN) An instantiated object containing the method to invoke

FN

(IN) Name of method in ’object’ that you want to invoke

Normal

(IN) Reserved

Return Value

A pointer to a UCX RV type, or NULL if an error occurs. The data type will depend upon the method/property that was invoked, and can be determined using UCXGetTypeFromRV().

Remarks

UCXCall provides an interface for C/C++ programs to access an objects’ methods. Before calling UCXCall(), you must create a parameter list which will be passed to the method being invoked.

See Also

"UCXCreateParam" on page 34
"UCXDestroyParam" on page 34
"UCXEZCall" on page 72

3.2.15 UCXCreateContext

Creates a context for use when calling UCX functions.

Syntax

```
void *UCXCreateContext (int ScreenID)
```

Parameters

ScreenID

(IN) Valid CLIB Screen ID (obtained by calling GetCurrentScreen()),or -1.

Return Value

Returns a pointer to an UCX Context, which is required to call all other UCX functions.

Remarks

UCX library functions use the client context to refer back to the calling process. If a client context holds a valid ScreenID, UCX functions can display messages on the calling processes console. Substitute -1 for a valid screen ID if you do not want UCX to display messages on your application console.

See Also

"UCXDestroyContext" on page 71
"UCXEZCall" on page 72

3.2.16 UCXDestroyContext

Frees a client context.

Syntax

```
void UCXDestroyContext (void *CP)
```

Parameters

CP

(IN) Context ID of caller

Return Value

None.

Remarks

Release memory associated with a client context. Also, when a context is destroyed, all active UCX libraries which have been accessed via the context are notified via an event, so they can cleanup any memory or other resources associated with the client.

See Also

"UCXCall" on page 70
"UCXCreateContext" on page 71

3.2.17 UCXEZCall

Allows C/C++ programs quick and easy access to UCX functions.

Syntax

```
void *UCXEZCall (void *CP, CHAR_PTR FN,...)
```

Parameters

CP

(IN) Context ID of caller

FN

(IN) UCX API Name that you want to execute

(Variable argument list)

Parameter type,value pairs

Return Value

Returns a pointer corresponding to the UCX RV type returned by the component else returns NULL if the system is out of memory.

Remarks

UCXEZCall provides a simple interface for C/C++ programs to access UCX functions, without having to construct UCX parameter lists. Instead, the function name is passed, along with argument type and value pairs, and UCXEZCall() does all the dirty work for you. Global objects can be accessed using UCXEZCall.

This API must be terminated with 2 NULL parameters.

See Also

"UCXCreateContext" on page 71
"UCXDestroyContext" on page 71
"UCXFreeRV" on page 76
"UCXGetTypeFromRV" on page 80

3.2.18 UCXGetArrayFromRV

Retrieves the value of an array from a UCX RV structure.

Syntax

```
void UCXGetArrayFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved array value else returns 0.

Remarks

Gets the array type from the UCX RETURN VALUE structure.

See Also

None.

3.2.19 UCXGetBooleanFromRV

Retrieves a boolean value from a UCX RV structure.

Syntax

```
uint32 UCXGetBooleanFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value if the retrieved boolean value is TRUE else returns 0.

Remarks

Gets the boolean type from the UCX RETURN VALUE structure.

See Also

None.

3.2.20 UCXGetByteFromRV

Retrieves a byte value from a UCX RV structure.

Syntax

```
uint8 UCXGetByteFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved byte value else returns 0.

Remarks

Gets the byte type from the UCX RETURN VALUE structure.

See Also

None.

3.2.21 UCXGetCurrencyFromRV

Retrieves a currency value from a UCX RV structure.

Syntax

```
sdouble UCXGetCurrencyFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved currency value else returns 0.

Remarks

Gets the currency type from the UCX RETURN VALUE structure.

See Also

None.

3.2.22 UCXGetDateFromRV

Retrieves a date value from a UCX RV structure.

Syntax

```
sdouble UCXGetDateFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved date value else returns 0.

Remarks

Gets the date type from the UCX RETURN VALUE structure.

See Also

None.

3.2.23 UCXGetDoubleFromRV

Retrieves a double value from a UCX RV structure.

Syntax

```
sdouble UCXGetDoubleFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved Dword value else returns 0.

Remarks

Gets the double type from the UCX RETURN VALUE structure.

See Also

None.

3.2.24 UCXGetDWordFromRV

Retrieves a Dword value from a UCX RV structure.

Syntax

```
uint32 UCXGetDWordFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved array value else returns 0.

Remarks

Gets the Dword type from the UCX RETURN VALUE structure.

See Also

None.

3.2.25 UCXFreeRV

Frees the memory associated with the return values.

Syntax

```
void UCXFreeRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

None.

Remarks

None.

See Also

None.

3.2.26 UCXGetIntegerFromRV

Retrieves a integer value from a UCX RV structure.

Syntax

```
sint16 UCXGetIntegerFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved integer value else returns 0.

Remarks

Gets the integer type from the UCX RETURN VALUE structure.

See Also

None.

3.2.27 UCXGetLongFromRV

Retrieves a long value from a UCX RV structure.

Syntax

```
sint32 UCXGetLongFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved long value else returns 0.

Remarks

Gets the long type from the UCX RETURN VALUE structure.

See Also

None.

3.2.28 UCXGetSingleFromRV

Retrieves a single value from a UCX RV structure.

Syntax

```
ssingle UCXGetSingleFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved single value else returns 0.

Remarks

Gets the single type from the UCX RETURN VALUE structure.

See Also

None.

3.2.29 UCXGetStringFromRV

Retrieves a string value from a UCX RV structure.

Syntax

```
puint8 UCXGetStringFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved string value else returns 0.

Remarks

Gets the string type from the UCX RETURN VALUE structure.

See Also

None.

3.2.30 UCXGetStringElementFromRV

Retrieves a value of the string element from a UCX RV structure.

Syntax

```
UCXStr UCXGetStringElementFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved string element value else returns 0.

Remarks

Gets the string type from the UCX RETURN VALUE structure.

See Also

None.

3.2.31 UCXGetStringLengthFromRV

Retrieves a length of the string from a UCX RV structure.

Syntax

```
uint32 UCXGetStringLengthFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved string length value else returns 0.

Remarks

Gets the numeric type from the UCX RETURN VALUE structure.

See Also

None.

3.2.32 UCXGetTypeFromRV

Retrieves a return type of the UCX RV structure.

Syntax

```
uint32 UCXGetTypeFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns the type of the return type else returns 0.

Remarks

Gets the type of the return type from the UCX RETURN VALUE structure.

See Also

None.

3.2.33 UCXGetUserTypeFromRV

Retrieves a type of the user from a UCX RV structure.

Syntax

```
void UCXRVUser (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved user value else returns 0.

Remarks

Gets the user type from the UCX RETURN VALUE structure.

See Also

None.

3.2.34 UCXGetWordFromRV

Retrieves a word value from a UCX RV structure.

Syntax

```
uint16 UCXGetWordFromRV (void *CP, void *UCXRV)
```

Parameters

CP

(IN) Context ID of caller

UCXRV

(IN) An UCX RETURN VALUE structure

Return Value

Returns a non-zero value corresponding to the retrieved word value else returns 0.

Remarks

Gets the word type from the UCX RETURN VALUE structure.

See Also

None.

3.2.35 UCXCopyContext

Makes an exact copy of a UCX Context Block.

Syntax

```
void *UCXCopyContext(void *old)
```

Parameters

old

(IN) Context that is to be copied

Return Value

Returns the UCX context handle if the specified context is copied else returns NULL if the system is out of memory

Remarks

None.

See Also

"UCXDeInitializeContext" on page 82
"UCXRestoreContext" on page 83

3.2.36 UCXDeInitializeContext

Deinitializes a UCX context handle.

Syntax

```
void *UCXDeInitializeContext(void *cp)
```

Parameters

CP

(IN) Context ID of caller

Return Value

Returns the context handle if it is deinitialized successfully else returns NULL if the system is out of memory.

Remarks

None.

See Also

"UCXRestoreContext" on page 83
"UCXCopyContext" on page 82

3.2.37 UCXRestoreContext

Restores required context members into original Context Handle.

Syntax

```
void *UCXRestoreContext(void *toCP, void *fromCP)
```

Parameters

toCP

(IN) Context ID of caller

fromCP

(IN) Context ID which is to be freed

Return Value

Remarks

Returns the value of toCP parameter if the context is restored successfully else returns NULL.

See Also

"UCXCopyContext" on page 82
"UCXDeInitializeContext" on page 82

3.2.38 UCXLoadClass

Loads a UCX class into memory.

Syntax

```
UCXClassLink *UCXLoadClass(void *CP, char *class)
```

Parameters

CP

(IN) Context ID of caller

class

(IN) The class name to be loaded

Return Value

Returns the pointer to the global link else returns NULL.

Remarks

This API is used to load any UCX class into memory. Once a class has been loaded, objects based on that class can be instantiated.

See Also

3.2.39 UCXCreateObject

Creates a UCX object data type.

Syntax

```
void *UCXCreateObject(void *CP, Char *name)
```

Parameters

CP

(IN) Context ID of caller

name

(IN) Function name that called this API

Return Value

Returns the pointer to the newly created object else returns NULL.

Remarks

An object created by this call is not bound to a class, and thus developers rarely use this call. It is provided for supporting the creation of EMPTY objects.

See Also

"UCXInstantiateObject" on page 84

3.2.40 UCXInstantiateObject

Instantiates an UCX object.

Syntax

```
void *UCXInstantiateObject(void *CP, char *Class, char *fn, void *param)
```

Parameters

CP

(IN) Context ID of caller

Class

(IN)Name of the UCX component

fn

(IN) Name of the calling function

Param

(IN) UCX Parameter to be passes while creating the object

Return Value

Returns the instantiated UCX object else returns NULL.

Remarks

This call is called to create an object

See Also

"UCXFreeRV" on page 76

3.2.41 UCXLocateObject

Returns a reference to the object.

Syntax

```
void *UCXLocateObject(GVP *CP, char *variable_name, int type)
```

Parameters

CP

(IN) Context ID of caller

variable_name

(IN) Name of the UCX global object

type

(IN) type of search

If the search type is 0 then the specific UCX global object is searched in the local variable list.

If the search type is 1 then the specific UCX global object is searched in the global variable list.

If the search type is 2 then the specific UCX global object is searched in both local and global variable lists.

Return Value

Returns the reference to the specific UCX global object else returns NULL.

Remarks

This call is used to search an already created object.

See Also

"UCXInstantiateGlobalObject" on page 86

3.2.42 UCXInstantiateGlobalObject

Creates an UCX global object.

Syntax

```
uint32 UCXInstantiateGlobalObject(void *CP, char *classname, char *varname,
char *FN)
```

Parameters

CP

(IN) Context ID of caller

classname

(IN) UCX component name

varname

(IN) Name of the newly created global object

FN

(IN) Name of the calling function

Return Value

Returns TRUE if the object is created successfully else returns FALSE.

Remarks

This call is called to create objects having global scope with in the CP. This call is used to create global scripting objects such as Err object. The objects created using this call can be invoked using UCXEZCall.

See Also

"UCXFreeRV" on page 76
"UCXInstantiateObject" on page 84

3.2.43 UCXGetDefaultItem

Returns the name of the default property or method for the component.

Syntax

```
char *UCXGetDefaultItem(void *CP, void *obj, void *params)
```

Parameters

CP

(IN) Context ID of caller

obj

(IN) Pointer to UCX object

params

(IN) UCX parameters to be passed during method invocation

Return Value

Returns the specified default property or the method name else returns NULL.

Remarks

UCX components can have default property or method. This API helps in retrieving the name of such default items. The client applications/scripting languages use this method to evaluate the abbreviated expression. For example Err is evaluated as Err.number.

3.3 Miscellaneous Functions

Miscellaneous functions are used by the components and the client for handling errors. The miscellaneous functions are:

UCXGetFatal
UCXSetFatal
UCXGetError
UCXSetError
UCXGetErrorText
UCXSetErrorText

3.3.1 UCXGetFatal

Retrieves the fatal error flag from a UCX context.

Syntax

```
uint32 UCXGetFatal (void *CP)
```

Parameters

CP

(IN) Context ID of caller

Return values

Returns 1 if the error flag represents TRUE else returns 0.

Remarks

None.

See Also

"UCXSetFatal" on page 88
"UCXGetError" on page 89

3.3.2 UCXSetFatal

Sets the fatal error flag in a UCX context.

Syntax

```
uint32 UCXSetFatal (void *CP, uint32 fatal_flag, sint32 error_code)
```

Parameters

CP

(IN) Context ID of caller

fatal_flag

(IN) 0 will set fatal flag to FALSE. Non-zero sets fatal flag to TRUE

error_code

(IN) The new error code value to set. See UCXSetError() for information

Return values

Returns the prior setting of the context's fatal error flag.

Remarks

None.

See Also

"UCXGetFatal" on page 87
"UCXSetError" on page 89
"UCXSetErrorText" on page 90

3.3.3 UCXGetError

Gets the error code in a UCX context.

Syntax

```
sint32 UCXGetError (void *CP)
```

Parameters

CP

(IN) Context ID of caller

Return values

Returns an integer value based on the error code of the specific UCX context else returns 0.

Remarks

None.

See Also

"UCXSetError" on page 89
"UCXGetErrorText" on page 90
"UCXSetErrorText" on page 90

3.3.4 UCXSetError

Sets the fatal error flag in a UCX context.

Syntax

```
sint32 UCXSetError (void *CP, sint32 error_code)
```

Parameters

CP

(IN) Context ID of caller

error_code

(IN) The new error code value to set. See UCXSetError() for information

Return values

Returns the prior setting of the context's fatal error flag.

Remarks

None.

See Also

"UCXGetError" on page 89
"UCXGetErrorText" on page 90
"UCXSetErrorText" on page 90

3.3.5 UCXGetErrorText

Gets the textual error information from a UCX context.

Syntax

```
CHAR_PTR UCXGetErrorText (void *CP)
```

Parameters

CP

(IN) Context ID of caller

Return values

Returns a pointer to the message that describes the last error that occurred.

Remarks

None.

See Also

"UCXGetError" on page 89
"UCXSetError" on page 89
"UCXSetErrorText" on page 90

3.3.6 UCXSetErrorText

Sets the textual error information in a UCX context.

Syntax

```
CHAR_TR UCXSetErrorText (void *CP,sint32 error_code,CHAR_PTR FN, CHAR_PTR Msg)
```

Parameters

CP

(IN) Context ID of caller

error_code

(IN) The new error code value to set. No lookup is performed.

FN

(IN) The function name. If this value is NULL it is ignored. Else, it is placed into the text message as follows: FN: MSG.

Msg

(IN) The textual representation of error_code

Return values

Returns the pointer to the new message text that has been saved else returns NULL if invalid parameters are passed.

Remarks

None.

See Also

"UCXGetError" on page 89
"UCXSetError" on page 89
"UCXGetErrorText" on page 90

Sample UCX Component

4

This section provides the basic details required during the development phase of an UCX component. It discusses about:

4.1 Sample Design Document

This section contains the design document for a sample UCX component.

4.1.1 Table of Contents

4.1.2 Introduction

This is the design of a Sample component to demonstrate the development of UCX components.

Component Description

This component allows you to get information about the company and employees of the company. Following NSP script illustrates the use of the component

```
<%
Set Camp = CreateObject("UCX:COMPANY")
Response.write Comp.Name

Set Amps = Comp.Employees
Set Emp = Emps.item(5)

'Display the details of 5th employee
Response.write Emp
Response.write Emp.id

'Display all employees
For each Emp in Emps
Response.write Emp.name
Next
%>
```

Component Methods

text goes here

COMPANY

This is the top-level object. This stores the information about company and allows to access employees of the company.

Constants

The following are the department related constants.

- ENGG- 1
- HR- 2
- ADMIN - 3

Properties

Data Type	Property Name	Attribute
String	Name	Read Only: Name of the company
String	Address	Read Write: Address of the company
EMPLOYEES	Employee	Read Only: Returns the Employees object on success else returns Null. This object is collection of the employees in the company.

EMPLOYEES

Allows to manage and reference Employee objects.

Properties

Data Type	Property Name	Attributes
Long	Count	Read Only: Number of employee in the company

Methods

Method Name	Parameter	Return Value
Item (Variant **Index**) This is the Default method for collections.	Index - The specified index of the employee object which is to be retrieved.	EMPLOYEE. Returns the Employee object if the specified employee is in the collection else returns NULL.
HasMoreElements ()	None.	Boolean. Returns TRUE if there are any more elements in the collection else returns FALSE.
Next ()	None.	EMPLOYEE. Returns the next Employee object from the collection else returns NULL.
Reset ()	None.	Void.

NOTE: : HasMoreElements, Next and Reset method are implemented because FOREACHNEXT function internally uses these methods.

EMPLOYEE

Retrieves the specific employee object from the Employees collection.

Properties

Data Type	Property Name	Attributes
String	Name	Read Only: Name of the employee. This is the Default property
Long	ID	Read Only: Employee Id of employee

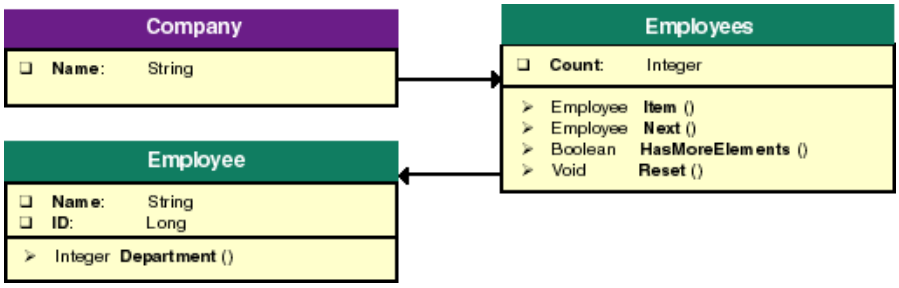
Methods

Method Name	Parameter	Return Value
ChangeDepartment ([Integer **NewDepartment**)	The ID of the new department to which the employee is to be changed.	Returns the previous department of employee. If no parameter is passed it returns the current department of employee.

Component Design

text goes here

Architecture diagram



Component Details

1. Name of the NLM: COMPANY.nlm
2. This NLM registers following UCX classes:
   - UCX:Company
   - UCX:Employees
   - UCX:Employee

General Information

This section includes any generic information such as error messages related to the component.

Error Messages

Error Code	Description
1	Error in instantiating the Object
2	Index out of range
3	Employee not found
4	Property cannot be set
5	Invalid parameter type

Error Code	Description
6	Invalid department name

4.2 Sample UCX Component

text goes here

4.2.1 Prerequisites for building UCX Components

CodeWarrior or Watcom compiler along with the NetWare NLM SDK files is required to build the UCX components.

4.2.2 Sample

This section provides a sample component to demonstrate the creation of UCX components.

The steps involved in creating an UCX component are

- Directory structure of the sample Source Files.
- Building the NLM using Build Instructions.
- Sample Design Document.
- Creating a Sample Component.
- Testing the Sample Component.

Source Files

The following directories are created during the installation.

1. SYS:NSN\SDK\IMP - Import files required for creation of UCX components.
2. SYS:NSN\SDK\INCLUDE - Include files required to build the UCX components.
3. SYS:NSN\SDK\C\CW - CodeWarrior project for the sample component.
4. SYS:NSN\SDK\C\WATCOM - Watcom make file for building the component using Watcom.
5. SYS:NSN\SDK\C\SRC - Sample component C source file and an ASP file which uses the component

Build Instructions

text goes here

CodeWarrior Build

Proceed with the following steps to build the component using CodeWarrior.

1. Open the CodeWarrior make file.
2. Set the paths properly to include and library directory.
3. Select Make.

Watcom Build

Proceed with the following steps to build the component using Watcom.

1. In the command (MS-DOS) prompt change the working directory to watcom.
2. Open the batch file BUILD.BAT and set the path to the directory in which the watcom binaries are present.
3. Execute the batch file.

NOTE: You can also build the NLM by executing the command wmake -f <make file name>

Creating a Sample Component

For creating an UCX component a clear component design document explaining the required properties, methods and sub objects of the component should be available.

Proceed with the following steps for creating an UCX component based on a specific design. The design of sample component is assumed for this purpose. Sample Design Document will provide more details on creating a design for a UCX component.

1. Create a C file and include the standard header files and the standard UCX macros.

```
#include <stdio.h>
#include <string.h>
#include "ucx.h"



UCXSTDPROCS();
UCXDEFINELIB();
UCXUNLOADERPROC();
```

   The macros, UCXSTDPROCS(), UCXDEFINELIB () and UCXUNLOADERPROC () define the required functions and the UCX component structure.

2. Define the parameters for the methods of the component.

```
// Parameter description for methods of Employees object
// Parameters for ITEM Method

UCXParameter EmployeesItemParams[] = {
// Param Type       Param Name
   {UCX_OBJECT_TYPE,     (CHAR_PTR)"Employee"},
   {UCX_ANY_TYPE,        (CHAR_PTR)"Index" },
   {NULL,                 NULL}
};

etc
```

3. Declare the method list.Organize the methods in the ascending order of the method name.

```
enum
{
 HASMOREELEMENTS,
   ITEM,
   NEXT,
   RESET,

};

// Employees  Methods

UCXMethod EmployeesMethods[] = {
//   Name of the method         Index(enum) of the method
Parameter block
    {(CHAR_PTR)"HASMOREELEMENTS",   HASMOREELEMENTS,
EmployeesHasMoreElementsParams},
    {(CHAR_PTR)"ITEM",         ITEM,         EmployeesItemParams},
    {(CHAR_PTR)"NEXT",       NEXT,   EmployeesNextParams},
(CHAR_PTR)"RESET", RESET, EmployeesResetParams},
    {NULL, NULL, NULL}
};
```

4 Declare the property list.

```
// Properties list of Company object

UCXProperty CompanyPropertyList[] = {
//Property Type Name of Property  Scope (PRIVATE/PUBLIC)
  { UCX_STRING_TYPE, CHAR_PTR)"NAME",UCX_PUBLIC },
  { UCX_STRING_TYPE, (CHAR_PTR)"ADDRESS", UCX_PUBLIC },
  { UCX_OBJECT_TYPE,  (CHAR_PTR)"EMPLOYEES",UCX_PUBLIC },
{ 0, NULL, 0 }

};
```

5 Declare the constant list.

```
// Constant list
UCXConstant CompanyConstantList[] = {
  { UCX_INTEGER_TYPE, (CHAR_PTR)"ENGG_DEPT",(CHAR_PTR)"1"},
  { UCX_INTEGER_TYPE, (CHAR_PTR)"HR_DEPT", (CHAR_PTR)"2"},
  { UCX_INTEGER_TYPE, (CHAR_PTR)"ADMIN_DEPT",
(CHAR_PTR)"3"},
  {0, NULL, NULL}
};
```

6 Declare the Events parameters and event list if the component raises events.

7 Repeat all the steps from Step 2 to Step 6 for all the sub objects.

8 Declare the class list.

```
//Class list

static UCXClass CompanyClassList[] =  {
{(CHAR_PTR)"UCX:COMPANY", 0, 0, 0, 0, &MyLib,NULL, NULL,
CompanyPropertyList, CompanyLibProc , (CHAR_PTR)"NAME"},
{(CHAR_PTR)"UCX:EMPLOYEES", 0, 0, 0, 0, &MyLib,
EmployeesMethods, NULL,EmployeesPropertyList,
EmployeesLibProc },
{(CHAR_PTR)"UCX:EMPLOYEE", 0, 0, 0, 0,
&MyLib,EmployeeMethods, NULL, EmployeePropertyList,
EmployeeLibProc , (CHAR_PTR)"NAME"},
{NULL}
};
```

9 Include the main function which takes care of registration of the component with the UCX library manager. Standard UCX macros can be used for this purpose.

```
// main block entry point

UCXENTRY(EMPLOYEE)
// Initialization and component registration is done here
{
   UCXINITLIBRARY2( CompanyClassList,
   (CHAR_PTR) "COMPANY",
   (CHAR_PTR) "Company Object",
   (CHAR_PTR) "----------------------------",
   (CHAR_PTR) "Copyright (c) 2000 Novell Inc. ,    All
    Rights Reserved.");

   UCXREGISTERLIBRARY();
}
```

10 Write the library procedure for the classes one by one. Include the code for performing various activities for getting and setting properties and invocation of methods.

```
static UCX_API CompanyLibProc(void *CP, void *params,
CHAR_PTR FN, uint32 Event, UCXClassLink *ClassLink, void
*Object)
{
  void  *Object_ptr = NULL ;

   switch( Event )
   {

  // Export constants. These constants can be used as script
constants.
    case UCX_EVENT_CONSTANT:
      return ( CompanyConstantList );
    break;

// Instantiate object
    case UCX_EVENT_OBJECT_CREATE:
     {

// Initialize the properties while creating the object.
// Return TRUE if object creation is success otherwise
return FALSE.
       UCXSetStringProperty(CP,Object,"NAME",COMPANY_NAME);
UCXSetStringProperty(CP,Object,"ADDRESS",COMPANY_ADDRESS
);
  return UCXCreateBoolean(CP, TRUE, FN);
     }
      break;

// Object getting destroyed. Do any necessary cleanup.
    case UCX_EVENT_OBJECT_DESTROY:

      break;

 // Get property value
    case  UCX_EVENT_PROPERTY_GET:

  // Identify the name of the property and perform the
action.
      if(strcmp ((const char *)FN , "EMPLOYEES") == 0)
    {

// Employees is the sub object. Create the sub object and
return the object.
// If required you can pass the arguments while creating
the sub objects
      if((Object_ptr = UCXInstantiateObject(CP,
"UCX:EMPLOYEES",NULL,NULL))==NULL)
         {
// Set the runtime error if the object creation fails.
UCXSetErrorText(CP, ERR_INSTAT, FN, (CHAR_PTR)"Error in
instantiating  Object");
        return  NULL;
         }
         else
         {

     return Object_ptr;
         }
```

```
     }

     break;

  // Set property value
  case UCX_EVENT_PROPERTY_SET:

    // Name property is read-only
    if ((strcmp ((const char *)FN ,"NAME") == 0) || (strcmp
((const char *)FN , "EMPLOYEES") == 0))
    {
      UCXSetErrorText(CP, ERR_PROPERTY_RO, FN,
(CHAR_PTR)"Property can not be set ");
      return NULL ;
    }

    // The default dispatch handler can set the Read write
properties.
// If setting the property implies an API call is being
made, which changes the system,
// then identify the property similar to above IF block
and then call the required API.
    break;
    }
    return(UCXDispatchEvent(CP, params, FN, Event,
ClassLink, Object));
}


Similarly handle the event,

    case UCX_EVENT_EXEC:

      // Identify the method index which helps in using a switch
statement
      nMethodIndex = UCXMethodIndex(CP, ClassLink->Class, FN);

      if( nMethodIndex == -1 )      /* didn't find the function */
        break;

      switch( nMethodIndex )
      {

// HasMoreElements method
        case HASMOREELEMENTS:

          UCXGetIntegerProperty(CP, Object, "INDEX", &nIndex);
          if ( nIndex < 10 )
            return UCXCreateBoolean(CP, TRUE, FN);
          else
            return UCXCreateBoolean(CP, FALSE, FN);
          break;

      }
```

Use UCXCreate<Data Type> functions to create required UCX return type and return the value from the library procedure.

Testing the Sample Component

Proceed with the following steps to test the sample component

1. Copy the NLM built using the Build Instructions to SYS:UCS\UCX directory on your NetWare server.
2. Add following line to SYS:\UCS\UCX.INI

   ```
   UCX:COMPANY=SYS:UCS\UCX\COMPANY.NLM
   ```
3. Copy the file COMPANY.ASP to web sever document directory. Ensure that execute bit is set for this file. Setting the execute bit can be achieved by using NetWare Enterprise Web server administration.
4. Verify the component by executing the ASP file, COMPANY.ASP from the browser.

About NSN Documentation

A

The NSN documentation comprises of 4 books.

1. The first book "NDK: Introduction to Novell Script for NetWare" contains the following chapters.
   - "Concepts" of NSN
   - "Tasks" - provides details about the tasks to be performed for installing NSN, and executing NSN and NSP scripts
   - "Reference Guide" - provides details about the UCX components supported by NSN.
   - "Legacy BASIC Statements" - provides details about the legacy BASIC supported by NSN.
   - "UCX Component Navigator" - includes a navigation table that gives the list of UCX components and the related objects.
2. The second book "NDK: NSN Components - Part One (A-L)" gives the details about the UCX components (A-L) and the related objects.
3. The third book "NDK: NSN Components - Part Two (N-Z)" gives the details about the UCX components (M-Z) and the related objects.
4. The fourth book gives the details about the UCX SDK API that are supported by NSN.

Quick Reference Index

B

This index provides the list of functions present in each category. Click on the specific function to get the relevant details.

B.1 Component API Index

text goes here

B.1.1 Component Builder Functions

UCXRegisterComponent
UCXDeregisterComponent
UCXGetClientClassLink
UCXMethodIndex
UCXDispatchEvent
UCXCreateParam
UCXDestroyParam
UCXGetArrayParam
UCXGetBooleanParam
UCXGetByteParam
UCXGetParamCount
UCXGetCurrencyParam
UCXGetDateParam
UCXGetDoubleParam
UCXGetDWordParam
UCXGetIntegerParam
UCXGetLongParam
UCXGetSingleParam
UCXGetRVPtrParam
UCXGetStringParam
UCXGetParamType
UCXGetWordParam

B.1.2 Parameter Functions

UCXCreateParam
UCXDestroyParam
UCXGetArrayParam
UCXGetBooleanParam
UCXGetByteParam
UCXGetParamCount
UCXGetCurrencyParam

B.1.3 Property Functions

B.2 Client API Index

B.3 Miscellaneous API Index

UCXGetError
UCXSetError
UCXGetErrorText
UCXSetErrorText

Revision History

C

The following table outlines all the changes that have been made to this documentation (in reverse chronological order):

C.1 June 2006

- Updated the Trademarks list to comply with revised Novell documentation standards

C.2 October 2001

- This book was existing as a part of the book Novell Script for NetWare (NSN) and has been split and re-organized
- Improved the contents for enhanced doc accessibility features