Object Interchange Service

Stanford Digital Library Project
Stanford University

July 21, 1995

Notices

Each of the following terms used in this publication is a trademark of another company:

OMG	Object Management Group
CORBA	Object Management Group
ORB	Object Management Group
IDL	Object Management Group

The term "CORBA" used throughout this publication refers to the Common Object Request Broker Architecture standards promulgated by the Object Management Group, Inc.

Table of Contents

Appendix A: Object Interchange Examples

This example of the DL Interchange protocol shows one implementation of a DL service serving out Dialog citations to a simple client using all asychronous communication. Other implementations works as well. For example, when we place object into one adress space in this examples below, they might instead be distributed. All objects can be in same or other address spaces, on local machine or remote.

The server proxy for Dialog only creates result collection objects and does not bother creating document objects. The client creates document objects.

1.1 Syntax:

The following examples use a python-like syntax to represent the arguments to methods and return results. All method arguments have their argument names associated with them in a record like format.

[] - an empty list (or sequence)

[1,2,3] - a list containing integers 1, 2, and 3.

[[], “the”, “an”] - a list containing an empty list, the words “the” and “an”.

{} - an empty record (with no fields)

{“aSize” : 20} - a record with one field “aSize” with a value of 20.

{“aSize” : 20, “aHandles” : [“a”, “b”, “c”]} - a record with two fields, “aSize” with a value of 20, and “aHandles” with a value of a list of words “a”, “b”, and “c”.

aResult - a denotes an instance object.

Result.AddItems(5) - Call the AddItems method on *Result* with an integer argument 5.

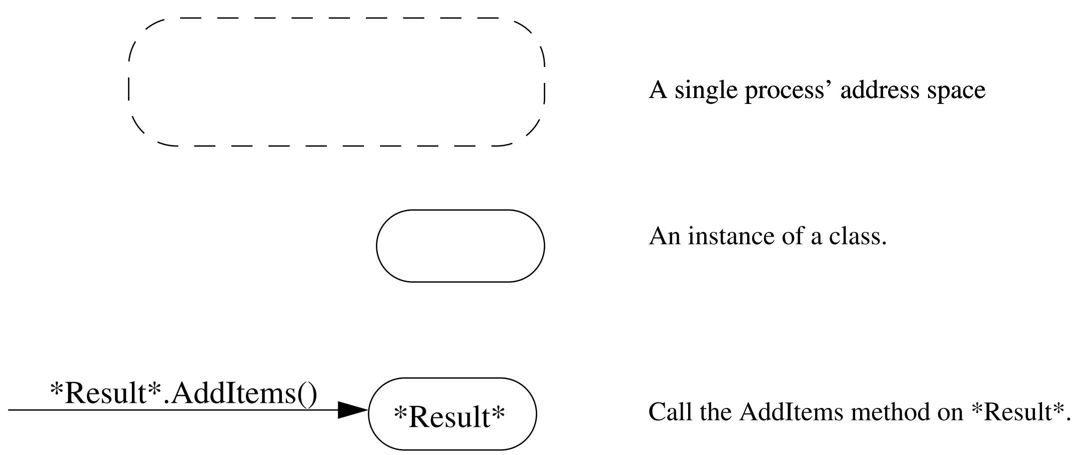
1.1.1 Symbol Prefixing

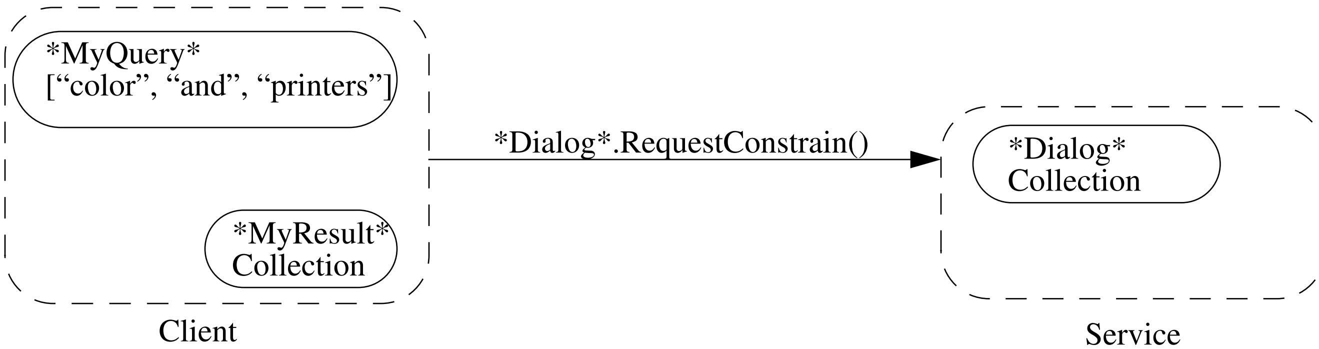
Parameter names, record field names, and class names used a type of prefixing. This is mainly used to determine the general type of the symbol independent of the context.

Table 1-1 Symbol prefixing

T	Non-class Type	k	constant
C	Class Interface	a	local variable
I	Interface Module	g	global variable
E	Enumeration Type	p	parameter to a method
		f	instance variable

‘CItem’ is a class, ‘pItems’ is a parameter to a method, ‘aItem’ is a local variable, and ‘fItem’ is an instance variable for an object instance.

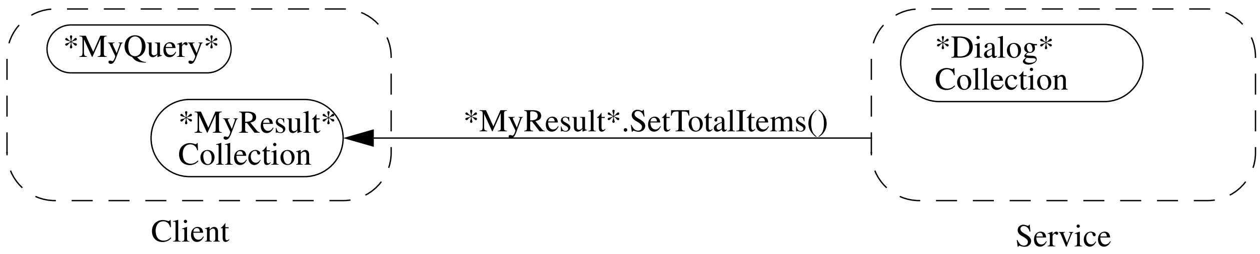




1. Request a search on Dialog for "color and printers" to be sent 10 documents with only the title and author to the awaiting collection *MyResult*.

RequestConstrain(
   pMessageID : 24601,
   pQuerySummary : {
      aQuery : *MyQuery*,
      aQueryComponents : ["color", "and", "printers"],
      aMoreSummary : []},
   pServicePrefSummary : {
      aServicePreferences: NULL,
      aNumberOfItems: 10,
      aItemProperties : ["title", "author"],
      aMoreSummary : []},
   pResultTarget : *MyResult*)

The search is performed on Dialog and 59 hits are returned. The Dialog Collection immediately calls SetTotalItems on *MyResult* to set the expected size of the result collection to 59.

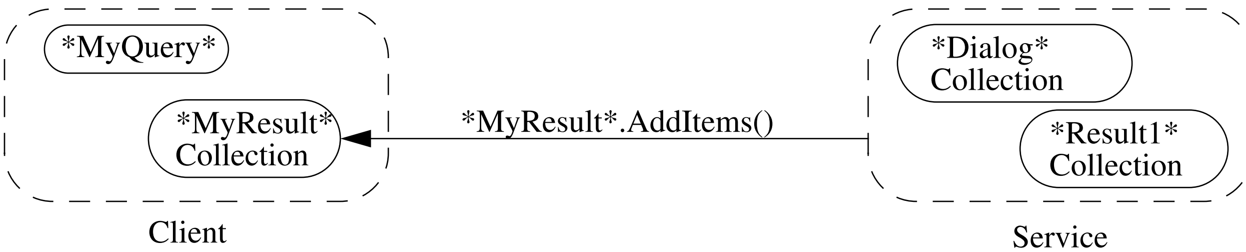


2. Set the total number of items in this collection.

MyResult.SetTotalItems(pNumItems : 59)

The client can now display the total number of items in the result set collection.

Now, a result collection is formed at the service side called *Result1* (Note: this is optional. The *Dialog* collection could choice to do all follow up work itself.) *Result1* requests from Dialog the first ten citations, parses them, and then calls AddItems:



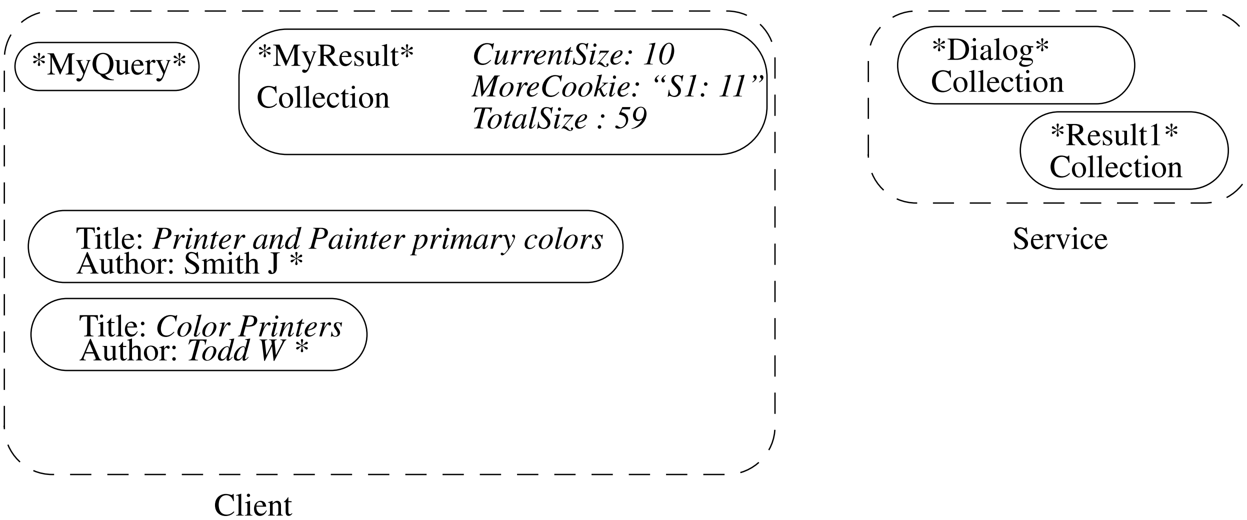
3. Send the 10 requested items to the client collection object along with a cookie “S1: 11” to pass to *Result1* when getting more documents using RequestItems(This could be done in one call as shown below, or in several seperate calls as the information becomes available from Dialog.)

**MyResult*.AddItems(pMessageID* : 24601,
pItems : {
aACs: [
[{*anItemClass* : “document”, *aTarget* : *Result1*, *aCookie* : “S1: 1”, *aHints*: []},
{*anItemClass* : “document”, *aTarget* : *Dialog*, *aCookie* : “an=3424601”, *aHints*: [] }],
[{*anItemClass* : “document”, *aTarget* : *Result1*, *aCookie* : “S1: 2”, *aHints*: []},
{*anItemClass* : “document”, *aTarget* : *Dialog*, *aCookie* : “an=4310642”, *aHints*: [] }],
... 8 more Access Capabilities ...],
aState : {
aNames : [“title”, “author”],
anItemsState : [
[“Printer and painter primary colors”, “Smith J”],
[“Color Printers”, “Todd W”],
... 8 more records ...] },
},
, *pNewMoreCookie*: [
[*anItemClass* : NULL, *aTarget* : *Result1*, *aCookie* : “S1: 11”, *aHints* : []],
[*anItemClass* : NULL, *aTarget* : *Dialog*, *aCookie* : “Query(color and printers),#11”, *aHints* : []]
])

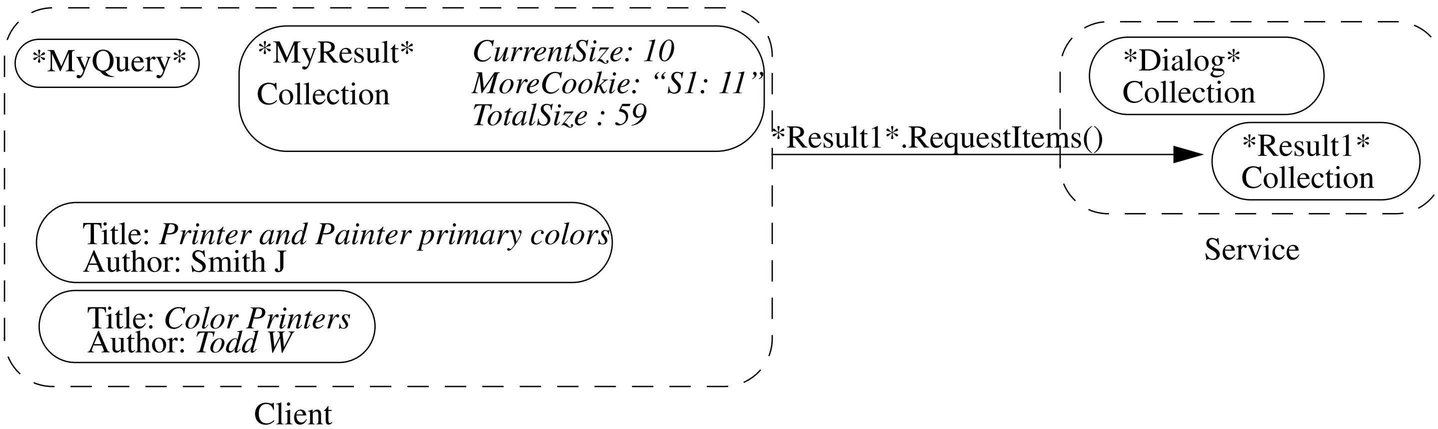
4. The server sends the CompleteRequest message to *MyResult* collection to signal that it is not going to be sending any more message concerning pMessageID's initial request.

**MyResult*.CompletedRequest*(*pMessageID* : 24601)

MyResult creates ten CItem objects and fills in the properties for Title and Author along with the Access-Capability according to the pItems parameters in AddItems. It remembers the whole MoreCookie Access Capability referring to “S1: 11”. It now can display this information in some interesting way to the user.



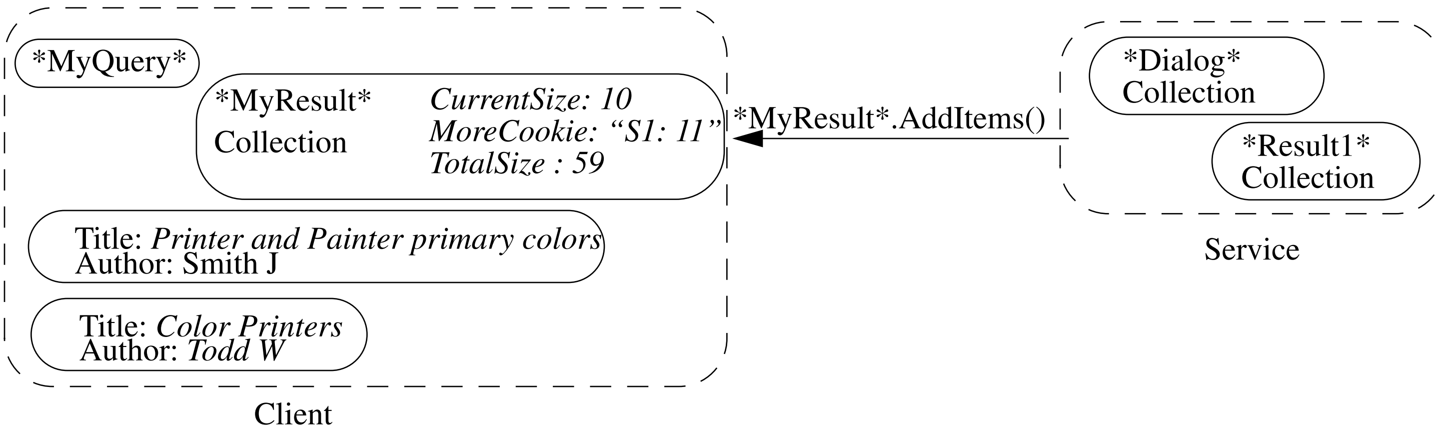
* Note: For figure clarity sake, the AccessCapability structure is not part of these figures. A CItem would store the AccessCapability structure associated with it.



5. When the next 20 items are needed:

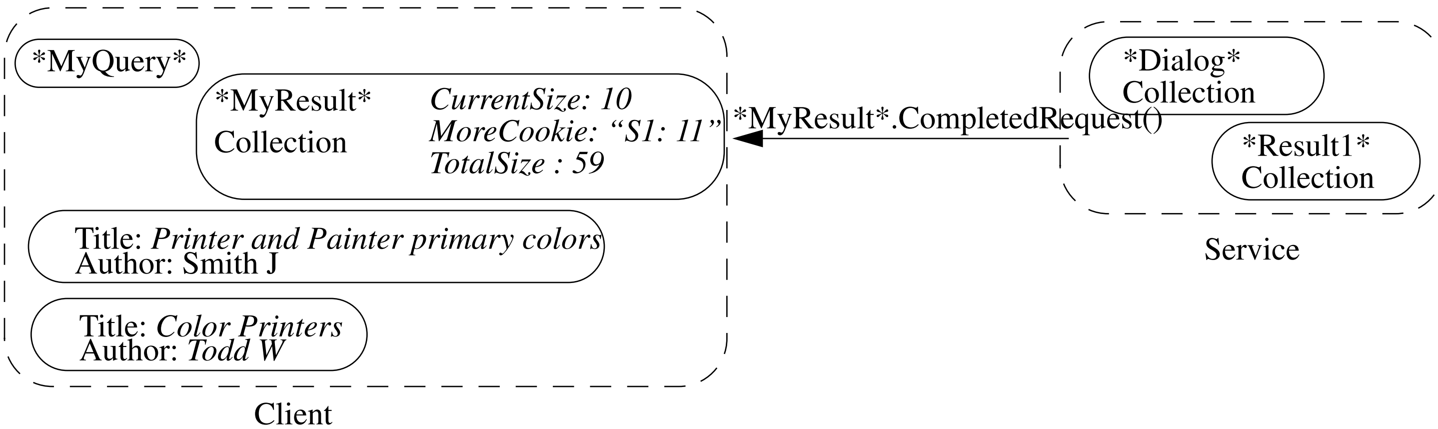
**Result1*.RequestItems*(
  pMessageID : 24602,
  pPropertyNames : [“title”, “author”],
  pMoreCookie : “S1: 11”,
  pNumberItems : 20,
  pResultTarget : *MyResult*
)

Result1 performs the request for 20 more items, parses them, and then calls AddItems() one or more times.



6. Send the 20 requested items to the client collection object along with a cookie "S1: 31" to pass to *Result1* when getting more documents using RequestItems.

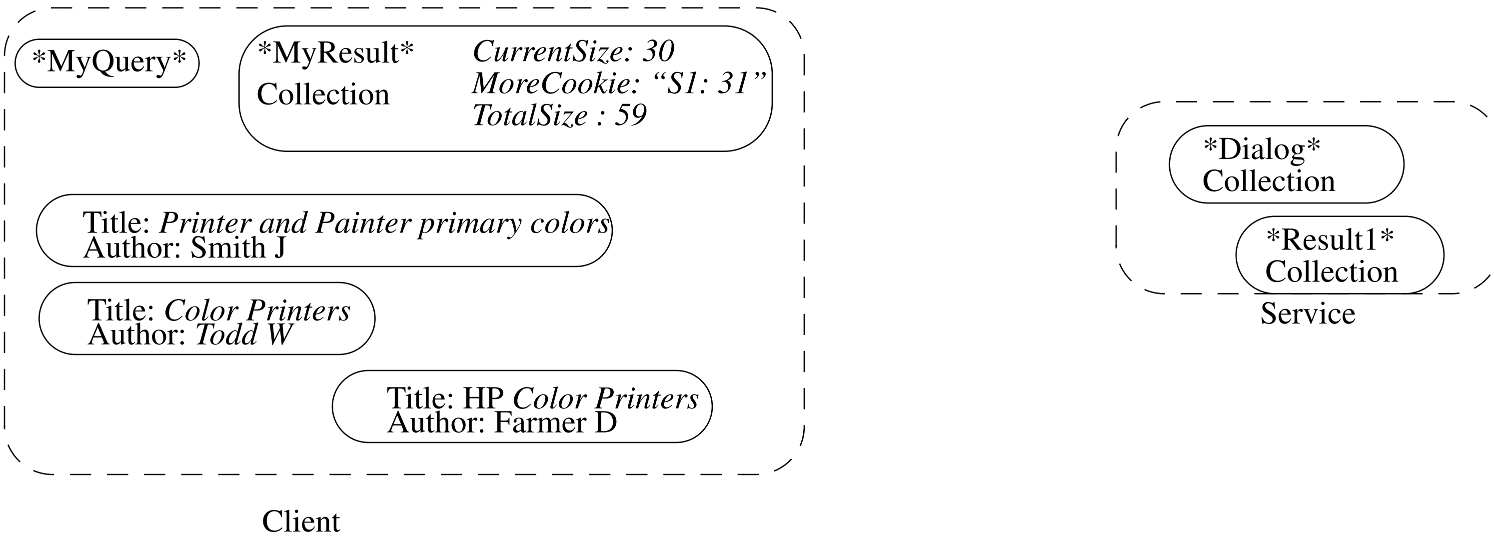
MyResult.AddItems(pMessageID : 24602,
    pItems : {
      aACs: [
      [{anItemClass : "document", aTarget : *Result1*, aCookie : "S1: 11", aHints: []},
          {anItemClass : "document", aTarget : *Dialog*, aCookie : "an=573638", aHints: [] }],
      ... 19 more Access Capabilities ...],
      aState : {
          aNames : ["title", "author"],
          anItemsState : [
              ["HP Color Printers", "Farmer D"],
              ... 19 more records ...] },
      },
    , pNewMoreCookie: [
      [anItemClass : NULL, aTarget : *Result1*, aCookie : "S1: 31", aHints : []],
      [anItemClass : NULL, aTarget : *Dialog*, aCookie : "Query(color and printers),#31", aHints : []]
      ])

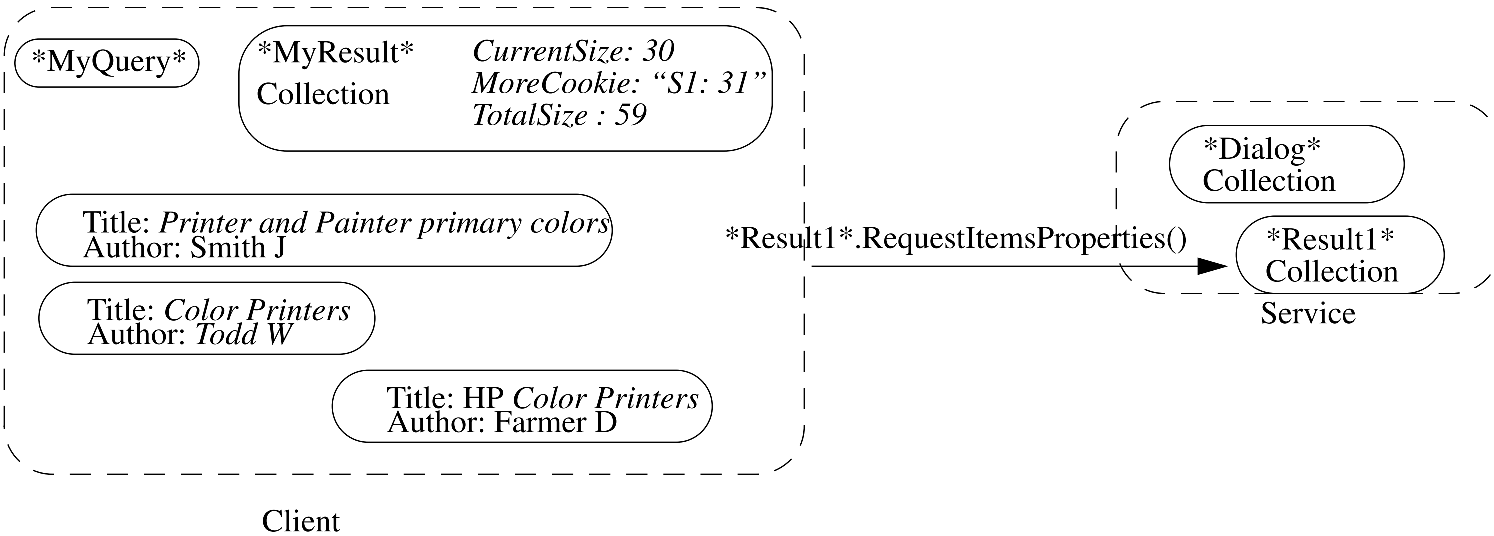


7. The server sends the CompleteRequest message to *MyResult* collection to signal that it is not going to be sending any more message concerning pMessageID’s initial request.

**MyResult*.CompletedRequest(pMessageID* : 24602)

MyResult creates twenty new CItem objects and fills in the properties for Title and Author along with the AccessCapability according to the pItems parameters in AddItems.

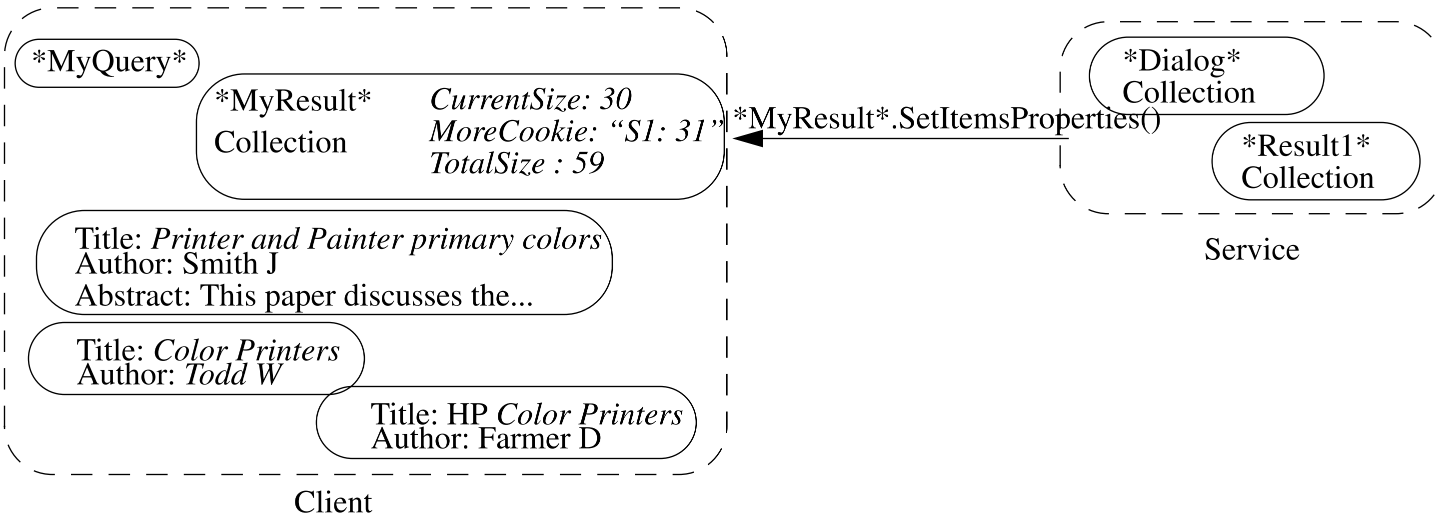




8. Now, the user wants the abstracts of items 1,5, and 9.

**Result1*.RequestItemsProperties*(
pMessageID : 24603,
pPropertyNames : [“abstract”],
pServerCookies : [“S1: 1”, “S1: 5”, “S1: 9”],
pClientCookies : [42, 43, 44],
pResultTarget : *MyResult*
)

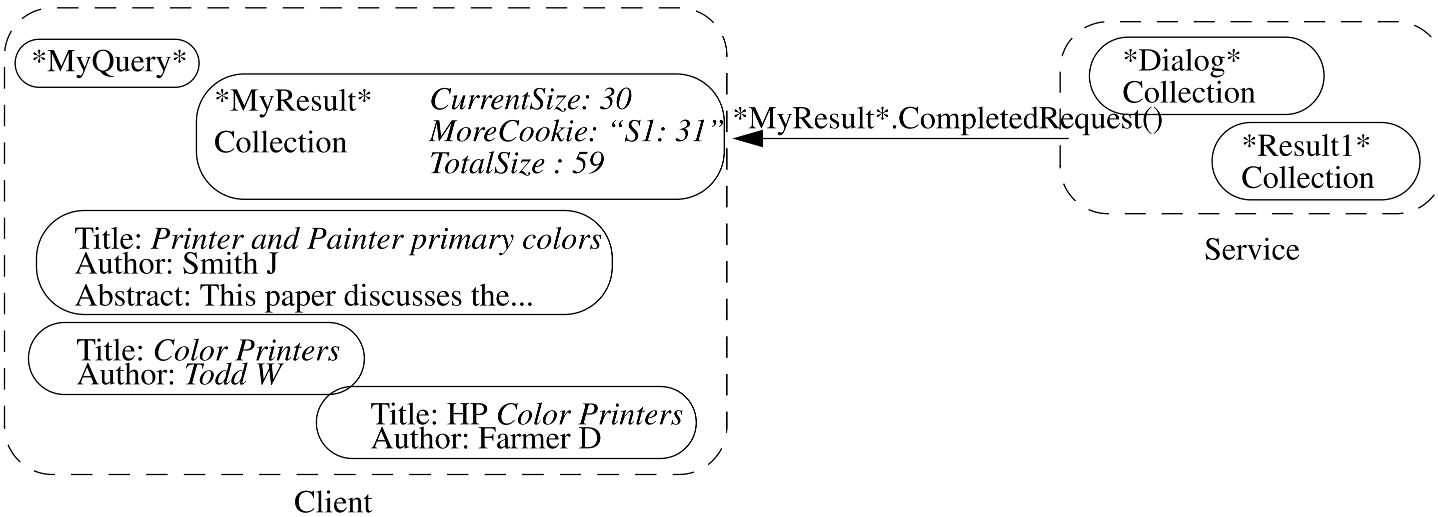
Result1 performs the request the abstracts of items 1,5, and 9, parses them, and then calls SetItemsProperties()



9. Send the 3 requested items' abstracts to the client collection object.

**MyResult*.SetItemsProperties*(
pMessageID : 24603,
pItems : {
aClientCookies : [42, 43, 44],
aState : {
aNames : ["abstract"],
anItemsState : [
["This paper discusses the basics of color and the relation to the systematics of painting..."],
... 2 more records ...] },
}
})

The client collection matches the client Cookie number of each anItemsState with an actual item object and sets the item's abstract property.



10. The server sends the CompleteRequest message to *MyResult* collection to signal that it is not going to be sending any more message concerning pMessageID’s initial request.

**MyResult*.CompletedRequest*(*pMessageID* : 24603)

Appendix B: Object Interchange Service ISL

B

The Object Interchange Service module defines the entire interchange service, consisting of data types, exceptions and interfaces described in previous sections.

```
(* ------------------------------------------------------------------------ *)
(* IDLInterchange.isl
                                                                            *)
(* ------------------------------------------------------------------------ *)
(* Symbol prefixing: I - Interface Module, C - Class Interface
              T - Non-class Type,    E - Enumeration Type
              k - constant           p - parameter to a method
              a - variable                                            *)
(* ------------------------------------------------------------------------ *)
(* Change Log:
    7/21/95 - removed ILifeCycle interface from CItem and CQuery.
            - Changed SUPERCLASS to SUPERTYPES.
    7/20/95 - Changed TItemHandle to TAccessOption
            - Changed TItemHandles to TAccessCapability
            - Changed TItemSetState to TItemsCookieState
            - Changed TItemAddState to TItemsACState
            - Changed TServicePreferences to TServicePrefSummary
            - Changed TQuery to TQuerySummary
            - Added CItem.CancelRequest() method.
            - Removed pMessageID from SetTotalItems
    7/ 6/95 - Added Exception Handling.
            - Changed NumberHits from CARDINAL to TOptionalSize.
            - Added methods RequestItemProperties and SetItemProperties
              to CItem. (probably should be in ICosProperty
    7/ 5/95 - Added MessageID's and Client Cookies.
            - Split ItemState into ItemSetState and ItemAddState.
*)
```

```
(* ---------------------------------------------------------------------- *)

INTERFACE IDLInterchange
     IMPORTS ICosProperty END;

(* ------------------------------------------------------------- *)
(* some simple types *)

TYPE TString = ilu.CString;

TYPE TOptionalSize = OPTIONAL CARDINAL;

TYPE Any = ICosProperty.Any;

(* ------------------------------------------------------------- *)
(* pre-declarations *)

TYPE PCItem = OPTIONAL CItem;

(* ------------------------------------------------------------- *)
(* define a cookie. *)

TYPE TCookie = Any;
TYPE TCookies = SEQUENCE OF TCookie;

(* ------------------------------------------------------------- *)
(* TMessageID is a client-side identifier for all the responses to
   a particular request. *)

TYPE TMessageID = TCookie;
```

```
(* ------------------------------------------------------------ *)
(* define Item Class *)

(* some item classes. *)
CONSTANT kItem : ilu.CString = "item";
CONSTANT kDocument : ilu.CString = "document"; (* inherits from item *)
CONSTANT kUSMarc : ilu.CString = "usmarc";      (* inherits from item *)


(* required properties defined on item *)
CONSTANT kItemContentType : ilu.CString = "Content-type";
CONSTANT kItemContentLength : ilu.CString = "Content-length";
CONSTANT kItemContent : ilu.CString = "Content";
CONSTANT kItemWeight : ilu.CString = "Weight";

(* some example properties defined on US Marc *)
CONSTANT kUSMarcTitle : ilu.CString = "ti";
CONSTANT kUSMarcAuthor : ilu.CString = "au";
CONSTANT kUSMarcAbstract : ilu.CString = "ab";
CONSTANT kUSMarcDate : ilu.CString = "da";

(* some example properties defined on document *)
CONSTANT kDocTitle : ilu.CString = "title";
CONSTANT kDocAuthor : ilu.CString = "author";
CONSTANT kDocAbstract : ilu.CString = "abstract";
CONSTANT kDocDate : ilu.CString = "date";

TYPE TItemClass = TString;
```

```
(* ------------------------------------------------------------ *)
(* define Access Capability *)

TYPE TAccessOption = RECORD
                       aTarget : PCItem,
                       aCookie : TCookie,
                       anItemClass : TItemClass,
                       aHints : ICosProperty.Properties
                     END;

TYPE TAccessCapability = SEQUENCE OF TAccessOption;

TYPE TAccessCapabilities = SEQUENCE OF TAccessCapability;
```

```
(* ------------------------------------------------------------- *)
(* Three types for transferring state for multiple items:
        1. Simplest case: only the state information is transmitted.
        2. Service provides access capabilities and associated state
           for multiple items.  Typical use: Initial response to a query.
        3. Client asked for more properties of items via their access
           capabilities, providing a set of associated client cookies.  The
           service now returns the requested properties, each with the
           associated client cookie.

   Note: this trades off interface beauty for minimizing information
         transfer.  :)

*)
TYPE TItemsState = RECORD
                      aNames : ICosProperty.PropertyNames,
                      anItemsState : ICosProperty.PropertyValuesList
                    END;
(* note: aNames and anItemsStates[i] must be of the same length. *)

TYPE TItemsCookieState = RECORD
                            aClientCookies : TCookies,
                            aStates : TItemsState
                          END;
(* note: aClientCookies and aStates.anItemsState must be of the same length. *)

TYPE TItemsACState = RECORD
                        aACs : TAccessCapabilities,
                        aState : TItemsState
                      END;
(* note: aACs and aStates.anItemsState must be of the same length. *)
```

```
(* ------------------------------------------------------------ *)
(* Exceptions - some possible exceptions.   Needs work.         *)

EXCEPTION InvalidRequest;
EXCEPTION InvalidQuery;

EXCEPTION InvalidProperties : ICosProperty.PropertyNames;
EXCEPTION InvalidMoreCookie;

EXCEPTION InvalidCookies : TCookies;

TYPE EExceptionReason = ENUMERATION
  kInvalidRequest,
  kInvalidQuery,
  kInvalidProperties,
  kInvalidMoreCookie,
  kInvalidCookies
END;

TYPE TException = EExceptionReason
  UNION
    aProperties: ICosProperty.PropertyNames = kInvalidProperties END,
    aCookies: TCookies = kInvalidCookies END
  END OTHERS;

TYPE TExceptions = SEQUENCE OF TException;

EXCEPTION MultipleExceptions : TExceptions;
```

```
(* ------------------------------------------------------------- *)
(* ############################################################# *)
(*
   CItem
*)
(* ############################################################# *)
(* ------------------------------------------------------------- *)

* ------------------------------------------------------------- *)
(* CItem holds all or part of the state of an information object. *)

TYPE CItem = OBJECT
  SUPERTYPES ICosProperty.PropertySet END
  METHODS
    (* return the access capability related to this item. *)
    GetAccessCapability() : TAccessCapability,

    (* add a list of access options to this item *)
    AddAccessOptions(pOptions : TAccessCapability),

    (* remove this list of access options from this item. *)
    RemoveAccessOptions(pOptions : TAccessCapability),

    (* ------------------------------------------------------------- *)
    (* RequestItemProperties()

       Request to be sent certain properties for this item.  Sometime
       later, one or more SetItemProperties messages will be sent to
       pResultTarget and finally a CompletedRequest() or RaiseException()
       will be sent to signal the end of the request.
         o pMessageID is the client-side data to pass back in order to
           identify the resulting callback messages.
         o pPropertyNames are the list of fields desired.
         o pResultTarget is the target result CItem.

       Note: this simply adds a asynchronous option to the ICosProperty
             interface: get properties from a single item.
    *)
    ASYNCHRONOUS
    RequestItemProperties(pMessageID : TMessageID,
                          pPropertyNames : ICosProperty.PropertyNames,
                          pResultTarget : CItem),

    (* ------------------------------------------------------------- *)
    (* SetItemProperties()

       Sets properties of this item.
         o pMessageID is the client-side data that matches the
           pMessageID of the initial request (a RequestItemProperties call)
```

```
     o pItem contains the state information requested.
*)
ASYNCHRONOUS
SetItemProperties(pMessageID : TMessageID,
                  pItem : ICosProperty.Properties),

(* ------------------------------------------------------------ *)
(* CompletedRequest()

   Signals to the client that the request with pMessageID
   has completed successfully.  No other messages with pMessageID
   shall be sent.
*)

ASYNCHRONOUS
CompletedRequest(pMessageID : TMessageID),

(* ------------------------------------------------------------ *)
(* RaiseException()

   Signals to the client that the request with pMessageID
   has encountered an exception.  No other messages with pMessageID
   shall be sent.
*)

ASYNCHRONOUS
RaiseException(pMessageID : TMessageID,
               pException : TException),

(* ------------------------------------------------------------ *)
(* CancelRequest()

   Signals to a requestee that the pending request with pMessageID
   shall be cancel.  No other messages with pMessageID shall be
   sent to the requestor.
*)

ASYNCHRONOUS
CancelRequest(pMessageID : TMessageID)

END;
```

```
(* ------------------------------------------------------------- *)
(* ############################################################# *)
(*
   CCollection
*)
(* ############################################################# *)
(* ------------------------------------------------------------- *)


(* CCollection contains a set of items (maybe documents) and
   efficently allows for the retrieval and setting of the items'
   properties in bulk.

   All of the Request methods are asynchronous in nature.  The results
   should come back sometime later with the SetItems or SetItemsProperties
   methods.

   The cookies are generated by the server collection and given to the
   client to hand back when more items are needed.  They may be used as
   an index into an array on the server side.  Other schemes are allowed.
   This protocol does not require items to be stored in a sequencial manner.
*)

TYPE CCollection = OBJECT
  SUPERTYPES CItem END
  METHODS
    (* ------------------------------------------------------------- *)
    (* GetTotalItems()

       Returns the total expected size of this collection.  If NULL,
       then the expected size is unknown.
    *)

    GetTotalItems() : TOptionalSize,

    (* ------------------------------------------------------------- *)
    (* SetTotalItems()

       Sets the total expected size of this collection.
         o pTotalSize is the expected total size of the collection.
           If NULL, then the expected size is unknown.

    *)

    ASYNCHRONOUS
    SetTotalItems(pTotalSize : TOptionalSize),

    (* ------------------------------------------------------------- *)
    (* GetItems()
```

```
   Get 'pNumberOfMoreItems' more Items.
     o pPropertyNames are the list of fields desired.
     o pNumberItems represents the maximum number of Items
       to send back.  If NULL, then return all of the items in
       this collection.
     o pMoreCookie was returned by AddItems that represents how to
       get more Items in the collection.  A value of NIL means to
       start from the beginning of the collection.
     o pNewMoreCookie is a new handle to get more Items.
*)

GetItems(pPropertyNames : ICosProperty.PropertyNames,
          pMoreCookie : TCookie,
          pNumberOfItems : TOptionalSize,
          OUT pNewMoreCookie : TAccessCapability
          ) : TItemsACState
 RAISES
      InvalidRequest,
      InvalidProperties, InvalidMoreCookie,
      MultipleExceptions
    END,

(* ------------------------------------------------------------ *)
(* RequestItems()

   Request to be sent pNumberItems more Items to pTarget.
   Sometime later, one or more AddItems messages will be sent to
   pResultTarget and finally a CompletedRequest() or RaiseException()
   will be sent to signal the end of the request.

     o pPropertyNames are the list of fields desired.
     o pMoreCookie was returned by AddItems that represents how to
       get more Items in the collection.  A value of NIL means to
       start from the beginning of the collection.
     o pNumberItems represents the maximum number of Items
       to send back.  If NULL, then return all of the items in
       this collection.
     o pTarget is the result collection to send the Items to.
*)

ASYNCHRONOUS
RequestItems(pMessageID : TMessageID,
             pPropertyNames : ICosProperty.PropertyNames,
             pMoreCookie : TCookie,
             pNumberItems : TOptionalSize,
             pResultTarget : CCollection),


(* ------------------------------------------------------------ *)
```

```
(* AddItems()

   Add additional Items to this collection.
     o pMessageID is the client-side data that matches the
       pMessageID of the initial request (a RequestConstrain or
       RequestItems call.)
     o pItems contains the state information requested.
     o pNewMoreCookie should be passed to collection to request
       more Items.  A NIL value means there are no more Items.
*)
ASYNCHRONOUS
AddItems(pMessageID : TMessageID,
          pItems : TItemsACState,
          pNewMoreCookie : TAccessCapability),


(* ---------------------------------------------------------- *)
(* GetItemsProperties()

   Get properties of items given their service-side cookies.

     o pPropertyNames are the list of fields desired.
     o pServerCookies is a list of server-side cookies of the items
       requested.

   Note: order and size of result TItemsState must match the
         order and size of the pServerCookies parameter.
*)

GetItemsProperties(pPropertyNames : ICosProperty.PropertyNames,
                    pServerCookies : TCookies
                    ) : TItemsState
 RAISES
      InvalidRequest,
      InvalidCookies, InvalidProperties,
      MultipleExceptions
    END,


(* ---------------------------------------------------------- *)
(* RequestItemsProperties()

   Request to be sent properties of items given their server-side
   cookies.

   Sometime later, one or more SetItemsProperties messages will be sent to
   pResultTarget and finally a CompletedRequest() or RaiseException()
   will be sent to signal the end of the request.

     o pMessageID is the client-side data to pass back in order to
```

```
          identify the resulting callback messages.
        o pPropertyNames are the list of fields desired.
        o pServerCookies is a list of server-side cookies of the items
          requested.
        o pClientCookies is a list of client-side cookies to be returned
          in conjuction with the items' state information.  The client can
          place anything in these cookies and is guarateed to be matched
          up with the resulting items.
        o pResultTarget is the target result collection to send the Items to.
  *)
  ASYNCHRONOUS
  RequestItemsProperties (pMessageID : TMessageID,
                          pPropertyNames : ICosProperty.PropertyNames,
                          pServerCookies : TCookies,
                          pClientCookies : TCookies,
                          pResultTarget : CCollection),

  (* ------------------------------------------------------------ *)
  (* SetItemsProperties()

     Set properties of existing Items in this collection.  If a item does not
     exist in this collection, raise an exception.
       o pMessageID is the client-side data that matches the
         pMessageID of the initial request (a RequestItemsProperties call)
       o pItems contains the state information requested.
  *)
  ASYNCHRONOUS
  SetItemsProperties (pMessageID : TMessageID,
                      pItems : TItemsCookieState),

  (* ------------------------------------------------------------ *)
  (* RaiseExceptions()

     Signals to the client that the request with pMessageID
     has encountered multiple exceptions.  No other messages with
     pMessageID should be sent.
  *)

  ASYNCHRONOUS
  RaiseExceptions (pMessageID : TMessageID,
                   pExceptions : TExceptions)


END;
```

```
(* ----------------------------------------------------------- *)
(* ########################################################### *)
(*
     CServicePreferences
*)
(* ########################################################### *)
(* ----------------------------------------------------------- *)

TYPE TServicePrefSummary = RECORD
             aServicePreferences : CServicePreferences,

             aNumberOfItems : TOptionalSize,
                  (* number of items to be initiallly returned. *)
                  (* if NULL, then the size means all available items *)


             aMoreSummary : ICosProperty.Properties
                  (* More service summary information to be sent along. *)
              END;

TYPE CServicePreferences = OBJECT OPTIONAL
  SUPERTYPES ICosProperty.PropertySet END
  METHODS
    GetSummary() : TServicePrefSummary,

    GetNumberOfItems() : TOptionalSize,
    SetNumberOfItems(pNumberOfItems : TOptionalSize),

END;
```

```
(* ------------------------------------------------------------ *)
(* ############################################################ *)
(*
     CQuery
*)
(* ############################################################ *)
(* ------------------------------------------------------------ *)

TYPE TQueryComponents = SEQUENCE OF TString;

TYPE TQuerySummary = RECORD
             aQuery : CQuery, (* the query object *)
             aQueryComponents : TQueryComponents,  (* the query string *)

             aItemPropertyNames : ICosProperty.PropertyNames,
                  (* Property names to be initially returned. *)

             aMoreSummary : ICosProperty.Properties
                  (* Morequery summary information to be sent along. *)
         END;

(* ------------------------------------------------------------ *)
(* CQuery represents the type, keywords, and other data needed to
   perform a constraint on a collection.
*)
TYPE CQuery = OBJECT OPTIONAL
  SUPERTYPES ICosProperty.PropertySet END
  METHODS
     GetSummary() : TQuerySummary,

     GetQueryComponents() : TQueryComponents,
     SetQueryComponents(pQueryComponents : TQueryComponents),

     GetItemPropertyNames() : ICosProperty.PropertyNames,
     SetItemPropertyNames(pItemPropertyNames : ICosProperty.PropertyNames)
END;
```

```
(* ------------------------------------------------------------ *)
(* ############################################################ *)
(*
     CConstrainCollection
*)
(* ############################################################ *)
(* ------------------------------------------------------------ *)

(* CConstrainCollection is a constrainable collection meaning that
   constraints can be placed on it (like a query or search.)  It is
   up to the implementation whether it wants to create a new collection
   to constrain or constrain this collection even further.  The pMoreCookie
   is returned to allow for both of these implementations.
*)

TYPE CConstrainCollection = OBJECT

  SUPERTYPES CCollection END

  METHODS
    (* ------------------------------------------------------------ *)
    (* Constrain()

       Constrain Collection with pDesignator and return pNumberOfItems Items
       back with properties of pPropertyNames.

         o pQuerySummary is the constraints to place on this collection.
         o pServicePrefSummary are miscellaneous preferences like quality of
               service, passwords, flow control, number of items to send now.
         o pTotalSize is the total expected size of this collection.  If
           NULL, then the total size is unknown at this time.
         o pMoreCookie should be passed to collection to request
           more Items.  A NIL value means there are no more Items.
    *)
    Constrain(pQuerySummary : TQuerySummary,
              pServicePrefSummary : TServicePrefSummary,
              OUT pTotalSize : TOptionalSize,
              OUT pMoreCookie : TAccessCapability) : TItemsACState
     RAISES
          InvalidRequest,
          InvalidQuery,
          MultipleExceptions
        END,

    (* ------------------------------------------------------------ *)
    (* RequestConstrain()

       Request to constrain Collection with pDesignator and send results
       to pTarget.
```

```
    Sometime later, one or more AddItems messages will be sent to
    pResultTarget and finally a CompletedRequest() or RaiseException()
    will be sent to signal the end of the request.

      o pMessageID is the client-side data to pass back in order to
        identify the resulting callback messages.
      o pQuerySummary is the constraints to place on this collection.
      o pServicePrefSummary are miscellaneous preferences like quality of
            service, passwords, flow control, number of itmes to send now.
      o pPropertyNames are the list of fields desired.
      o pTarget is the result collection to send the Items to.

  *)
  ASYNCHRONOUS
  RequestConstrain(pMessageID : TMessageID,
                   pQuerySummary : TQuerySummary,
                   pServicePrefSummary : TServicePrefSummary,
                   pResultTarget : CCollection)


END;
```

Appendix C: Object Property Service ISL

```
(* ---------------------------------------------------------------------- *)
(* Common Object Service
                                                                          *)
(* ---------------------------------------------------------------------- *)
(* ICosProperty - Interface Module for the Common Object Service
                  "Object Property Service"

                - Modeled after OMG's Object Property Service
                  specification.  OMG TC Document 95.6.1
                                                                          *)
(* ---------------------------------------------------------------------- *)

INTERFACE ICosProperty;

(* ---------------------------------------------------------------------- *)
(* some simple types *)

TYPE TString = OPTIONAL ilu.CString;

TYPE TObject = RECORD
                  aSBH : TString,
                  aType : TString
                END;

(* ---------------------------------------------------------------------- *)
(* The Any type.  ILU should really have this concept.
     Note: this probably should be in another Interface.
*)

TYPE TAnyList = SEQUENCE of Any;

TYPE AnyUnion = UNION
```

```
    INTEGER, CARDINAL, BYTE, BOOLEAN, REAL, CHARACTER,
    TObject,
    TAnyList,
    TString
  END;

TYPE Any = OPTIONAL AnyUnion;

(* ---------------------------------------------------------------- *)
(* property data types *)

TYPE PropertyName = TString;

TYPE Property = RECORD
  property-name : PropertyName,
  property-value : Any
END;

TYPE PropertyNames = SEQUENCE OF PropertyName;
TYPE Properties = SEQUENCE OF Property;

TYPE PropertyValues = SEQUENCE OF Any; (* SWH 7.20.95 *)
TYPE PropertyValuesList = SEQUENCE OF PropertyValues; (* SWH 7.20.95 *)

TYPE PropertiesList = SEQUENCE OF Properties;  (* SWH 6.15.95 *)

(* ---------------------------------------------------------------- *)
(* exceptions *)

EXCEPTION ConstraintNotSupported;
EXCEPTION InvalidPropertyName;
EXCEPTION ConflictingProperty;
EXCEPTION PropertyNotFound;
EXCEPTION UnsupportedTypeCode;
EXCEPTION UnsupportedProperty;
EXCEPTION UnsupportedMode;
EXCEPTION FixedProperty;
EXCEPTION ReadOnlyProperty;

TYPE ExceptionReason = ENUMERATION
  invalid-property-name,
  conflicting-property,
  property-not-found,
  unsupported-type-code,
  unsupported-property,
  unsupported-mode,
  fixed-property,
  read-only-property
END;
```

```
TYPE PropertyException = RECORD
  reason : ExceptionReason,
  failing-property-name : PropertyName
END;

TYPE PropertyExceptions = SEQUENCE OF PropertyException;

TYPE MultipleExceptions = RECORD
  exceptions : PropertyExceptions
END;

EXCEPTION MultipleExceptions : MultipleExceptions;

(* ------------------------------------------------------------ *)

TYPE PropertySet = OBJECT OPTIONAL
  METHODS
    define-property (property-name : PropertyName,
                     property-value : Any)
      RAISES InvalidPropertyName, ConflictingProperty, UnsupportedTypeCode,
             UnsupportedProperty, ReadOnlyProperty END,

    define-properties (nproperties : Properties)
      RAISES MultipleExceptions END,

    get-number-of-properties () : CARDINAL,

    get-all-property-names (how-many : CARDINAL,
                            OUT property-names : PropertyNames,
                            OUT rest : PropertyNamesIterator),

    get-property-value (property-name : PropertyName) : Any
      RAISES PropertyNotFound, InvalidPropertyName END,

    get-properties (property-names : PropertyNames,
                    OUT nproperties : Properties) : BOOLEAN,

    get-all-properties (how-many : CARDINAL,
                        OUT nproperties : Properties,
                        OUT rest : PropertiesIterator),

    delete-property (property-name : PropertyName)
      RAISES PropertyNotFound, InvalidPropertyName, FixedProperty END,

    delete-properties (property-names : PropertyNames)
      RAISES MultipleExceptions END,

    delete-all-properties () : BOOLEAN,

    is-property-defined (property-name : PropertyName) : BOOLEAN
```

```
      RAISES InvalidPropertyName END
  END;

(* ------------------------------------------------------ *)
(* used with get-all-property-names method *)

TYPE PropertyNamesIterator = OBJECT OPTIONAL
  METHODS
    reset (),
    next-one (OUT property-name : PropertyName) : BOOLEAN,
    next-n (how-many : CARDINAL, OUT property-names : PropertyNames) : BOOLEAN,
    destroy ()
  END;

(* ------------------------------------------------------ *)
(* used with get-all-properties method *)

TYPE PropertiesIterator = OBJECT OPTIONAL
  METHODS
    reset (),
    next-one (OUT aproperty : Property) : BOOLEAN,
    next-n (how-many : CARDINAL, OUT nproperties : Properties) : BOOLEAN,
    destroy ()
  END;

(* ------------------------------------------------------ *)
```