

USIGS Common Object Specification

*National Imagery and Mapping Agency
United States Imagery and Geospatial Service*

Version 1.5.1a

6 August 2001 to 15 Oct 2001

Acknowledgments

Many individuals and organizations provided support and technical contributions to this work. Individuals from numerous government agencies, contractor organizations and vendors contributed significantly to the development of this specification. We acknowledge these contributions and hope that these individuals and organizations will continue to actively support future updates and extensions. Thanks in advance.

Revision History

- Initial release of USIGS Common Object Specification (UCOS). Derived from earlier work on Imagery Access Services and Geospatial and Imagery Access Services Version 1.0 - Released for NCCB submittal 22 July 1997
- Revision of UCOS. Update to incorporate responses and comments from additional interface prototyping tests. Version 1.1 - Released for NCCB submittal 4 February. 1998
- Revision of UCOS. Update to incorporate responses and comments. Version 1.2 - Released for NCCB submittal 17 April. 1998. Version 1.2 – Released for NCCB submittal on 2 July 1998.
- Revision of UCOS. Update to incorporate responses and comments. Version 1.2 - Released on 2 October as part of RFC N01-0085 for NCCB review on 24 November.
- Release UCOS 1.3, dated 9 November 1998. Includes changes and mods resulting from UIP WG, 3-4 November. Changes included in Mod Package to RFC N01-0085, dated 9 November 1998.
- UCOS Version 1.3. Approved on 24 November at NCCB. As part of RFC N01-0085. Approval date annotated to document.
- UCOS Version 1.3. Approved on 22 June at NCCB. As part of RFC N01-0114. E2.0 As Built Baseline.
- UCOS Version 1.4. Draft released on 4 June as part of RFC N01-0127 for UIP Baseline Update for E2.5. Update released on 5 August based on results of 3-4 August UIP/API WG for inclusion as part of Mod Package for RFC N01-0127.
- UCOS Version 1.5. Final Draft released on 18 February 2000 as part of RFC N01-0148 for UIP Baseline Update for the NE028/NE022 era. Update released on 18 February 2000 based on results of 19 January 2000 UIP/API WG.
- RFC N01-0148 withdrawn. UCOS Version 1.5. Final Draft re-released on 21 April 2000 as part of RFC N01-0203 for UIP Baseline Update for NE049/NE028/NE022 effectivities.
- UCOS V1.5, Final release dated 26 June 2000. Approved by NCCB on 26 June 2000 as part of RFC N01-0203J.
- UCOS V1.5.1 Final release dated 6 August 2001. Approved by NCCB on 6 August 2001 as part of RFC N01-0268.

Planned Releases

- Regular updates at approximately six month intervals or as required to support additional interface definition efforts.

UNCLASSIFIED

Preface

This document defines common interfaces and data types that are expected to be used by many other United States Imagery and Geospatial Service (USIGS) interface specifications. The intent of this specification is to document the interfaces, data types and error conditions that are expected to most commonly occur or be most broadly applicable across the USIGS architecture. The use of these common definitions will support interoperability among the various interface specifications in the USIGS architecture.

This specification was prepared consistent with industry practices and is modeled after those being prepared by the Object Management Group (OMG) industry consortium. This approach is also consistent with guidelines and direction established by NIMA through its Architecture and Standards processes.

Table of Contents

1. OVERVIEW	1
2. UCO ELEMENTS.....	2
2.1. MODULE UCO.....	2
2.2. UCO DATA TYPES.....	2
2.2.1. UCO General Data Types.....	2
2.2.1.1. Name, NameName, NameNameList.....	2
2.2.1.2. NameList, NameValue, NameValueList.....	2
2.2.1.3. FileLocation, Protocol, DirectAccessLocation, FileLocationList, and FileSize (j/NPS).....	3
2.2.1.4. Percentage, Angle, Ratio, DateRate, and UCOS NULL:.....	4
2.2.1.5. Date, Time, TimeDuration, and AbsTime	5
2.2.1.6. EmailAddress	6
2.2.1.7. DAG, DAGList, NodeID, NodeType, Node, NodeList, Cardinality, Edge, and EdgeList.....	6
2.2.1.8. StringDAG, StringDAGList.StringNode, StringNodeList	8
2.2.1.9. ChangeType, Change, ChangeList, UpdateDAG and UpdateDAGList	9
2.2.1.10. RequestDescription and RequestDescriptionList (j/NPS).....	9
2.2.1.11. BinData	10
2.2.1.12. XMLDocument	10
2.2.1.13. EntityGraph, EntityNode, EntityRelationship, EntityNodeList, and EntityRelationshipList	10
2.2.2. Vector-like Data Types	11
2.2.2.1. Coordinate2d and Coordinate3d.....	11
2.2.2.2. LineString2d and LineString3d	12
2.2.2.3. Polygon and PolygonSet	12
2.2.2.4. Rectangle, RectangleList, LinearDimension, AreaDimension, Height, Elevation, Distance, Radius, Angle, Circle, Ellipse, and Weight	12
2.2.3. Raster-like Data Types	14
2.2.3.1. BufferType and Buffer	14
2.2.3.2. SimpleGSIImage	15
2.2.3.3. SimpleCImage and CompressedImage	15
2.2.4. State and Status (j/NPS).....	16
2.3. UCO INTERFACES	17
2.4. UCO EXCEPTIONS (j/NPS).....	17
2.4.1. Exception details (j/NPS)	18
2.4.2. InvalidInputParameter (j/NPS)	18
2.4.3. ProcessingFault (j/NPS)	19
2.4.4. SystemFault (j/NPS)	20
3. PROFILE SERVICE ELEMENTS	21
3.1. MODULE PS	21
3.2. PS DATA TYPES	21
3.2.1. ProfileElementType, ProfileElementTypeList	21
3.3. PS INTERFACES	21
3.3.1. Interface Profile	21
3.3.1.1. get profile element types	22
3.3.1.2. get profile element	23
3.3.1.3. get secure profile element	23
3.3.1.4. get secure view profile element	23
3.3.1.5. get profiled views	24
3.3.1.6. get last update time	24
3.3.2. Interface ProfileMgr	24

3.3.2.1. get profile	25
3.3.2.2. get version	25
3.3.2.3. set user password.....	25
3.3.3. Interface ProfileElement	26
3.3.3.1. get last update time	26
3.3.4. Not Used	26
3.3.5. Interface BasicProfile	26
3.3.5.1. User Information	29
3.3.5.2. Security Information	29
3.3.5.3. User Preferences	29
3.3.6. Interface SecureProfile	30
3.3.6.1. get authorization	32
3.3.6.2. get authorization classification	32
3.3.6.3. get exception by countries	33
3.3.6.4. get releasable countries	33
3.3.6.5. get releasable organizations	33
3.3.6.6. get distribution limitation codes	33
3.3.6.7. get access agreements	33
3.3.6.8. get compartments	34
3.3.6.9. get restricted attributes	34
3.3.6.10. get restricted entities	34
3.3.7. Interface SecureViewProfile	34
3.3.7.1. use authorization	35
3.3.7.2. get restricted attribute values	35
3.3.8. Interface GIASProfile	35
3.3.8.1. get allowable operations	49
3.3.8.2. operation is allowed	49
3.3.8.3. list volumes	50
3.3.8.4. new folder	50
3.3.8.5. update folder	50
3.3.8.6. remove folder	50
3.3.8.7. list folders	50
3.3.8.8. update folder item	51
3.3.8.9. get entry	51
3.3.8.10. remove entry	51
3.3.8.11. list entries	52
3.3.8.12. list entry items	52
3.3.8.13. new submitted query	52
3.3.8.13. new submitted query and hitcount	53
3.3.8.14. new submitted hitcount	53
3.3.8.15. new submitted order	53
3.3.8.16. new submitted create	54
3.3.8.17. new standing query	54
3.3.8.18. new standing order	54
3.3.8.19. new saved query	55
3.3.8.20. update saved query	55
3.3.8.21. new saved order	56
3.3.8.22. update saved order	56
3.3.8.23. new results digest	57
3.3.8.24. update results digest	57
3.3.8.25. get results digest	58
3.3.8.26. new saved session	58
3.3.8.27. update saved session	58

3.3.8.28. get saved session.....	59
3.3.9. Data Types.....	59
3.3.9.1. SubmittedQuery and SubmittedQueryList	59
3.3.9.1. SubmittedQueryAndHitCount and SubmittedQueryAndHitCountList.....	59
3.3.9.2. SubmittedOrder and SubmittedOrderList	60
3.3.9.3. SubmittedCreate and SubmittedCreateList	60
3.3.9.4. SubmittedHitCount and SubmittedHitCountList	60
3.3.9.5. StandingQuery and StandingQueryList.....	61
3.3.9.6. StandingOrder and StandingOrderList.....	61
3.3.9.7. SavedQuery and SavedQueryList	61
3.3.9.8. SavedOrder and SavedOrderList	62
3.3.9.9. ResultsDigest and ResultsDigestList.....	62
3.3.9.10. SavedSession and SavedSessionList.....	63
3.4. PS EXCEPTIONS.....	63
3.4.1. UnknownProfileElementType.....	64
3.4.2. BadProfileElement.....	64
3.4.3. BadAccessCriteria.....	64
3.4.4. BadAccessValue.....	64
3.4.5. PasswordExpired	64
3.4.6. InvalidOrder	65
3.4.7. InvalidQuery.....	65
3.4.8. UnknownOperation.....	65
3.4.9. VolumeReadAccessDenied	65
3.4.10. VolumeWriteAccessDenied	65
3.4.11. FolderNotEmpty	65
3.4.12. FolderExists.....	66
3.4.13. UnknownFolderItemType	66
3.4.14. UnknownVolume	66
3.4.15. UnknownPRID	66
3.4.16. PermissionDenied	66
3.4.17. BadFileLocation	66
3.4.18. SystemFault	66
5. UNIVERSAL PRODUCT IDENTIFIER.....	68
5.1. UID	68
APPENDIX A: UCO IDL	69
APPENDIX D: UNIVERSAL PRODUCT IDENTIFICATION.....	116
APPENDIX E: ENCODING RULES FOR STRINGDAG.....	117
APPENDIX F: REFERENCE OMG STANDARD IDL.....	120
APPENDIX G: ACRONYMS	1
APPENDIX H: POINTS OF CONTACT	2
1. OVERVIEW	4
2. UCO ELEMENTS	4
2.1. MODULE UCO	4
2.2. UCO DATA TYPES	4

<u>2.2.1. UCO General Data Types</u>	4
2.2.1.1. Name, NameName, NameNameList	4
2.2.1.2. NameList, NameValue, NameValueList	4
2.2.1.3. FileLocation, Protocol, DirectAccessLocation, FileLocationList, and FileSize (j/NPS)	4
2.2.1.4. Percentage, Angle, Ratio, DateRate, and UCOS_NULL;	4
2.2.1.5. Date, Time, TimeDuration, and AbsTime	4
2.2.1.6. EmailAddress	4
2.2.1.7. DAG, DAGList, NodeID, NodeType, Node, NodeList, Cardinality, Edge, and EdgeList	4
2.2.1.8. StringDAG, StringDAGList, StringNode, StringNodeList	4
2.2.1.9. ChangeType, Change, ChangeList, UpdateDAG and UpdateDAGList	4
2.2.1.10. RequestDescription and RequestDescriptionList (j/NPS)	4
2.2.1.11. BinData	4
2.2.1.12. XMLDocument	4
2.2.1.13. EntityGraph, EntityNode, EntityRelationship, EntityNodeList, and EntityRelationshipList	4
<u>2.2.2. Vector like Data Types</u>	4
2.2.2.1. Coordinate2d and Coordinate3d	4
2.2.2.2. LineString2d and LineString3d	4
2.2.2.3. Polygon and PolygonSet	4
2.2.2.4. Rectangle, RectangleList, LinearDimension, AreaDimension, Height, Elevation, Distance, Radius, Angle, Circle, Ellipse, and Weight	4
<u>2.2.3. Raster like Data Types</u>	4
2.2.3.1. BufferType and Buffer	4
2.2.3.2. SimpleGSImage	4
2.2.3.3. SimpleCImage and CompressedImage	4
<u>2.2.4. State and Status (j/NPS)</u>	4
<u>2.3. UCO INTERFACES</u>	4
<u>2.4. UCO EXCEPTIONS (j/NPS)</u>	4
2.4.1. Exception_details (j/NPS)	4
2.4.2. InvalidInputParameter (j/NPS)	4
2.4.3. ProcessingFault (j/NPS)	4
2.4.4. SystemFault (j/NPS)	4
<u>3. PROFILE SERVICE ELEMENTS</u>	4
<u>3.1. MODULE PS</u>	4
<u>3.2. PS DATA TYPES</u>	4
3.2.1. ProfileElementType, ProfileElementTypeList	4
<u>3.3. PS INTERFACES</u>	4
3.3.1. InterfaceProfile	4
3.3.1.1. get_profile_element_types	4
3.3.1.2. get_profile_element	4
3.3.1.3. get_secure_profile_element	4
3.3.1.4. get_secure_view_profile_element	4
3.3.1.5. get_profiled_views	4
3.3.1.6. get_last_update_time	4
3.3.2. InterfaceProfileMgr	4
3.3.2.1. get_profile	4
3.3.2.2. get_version	4
3.3.2.3. set_user_password	4
3.3.3. InterfaceProfileElement	4
3.3.3.1. get_last_update_time	4
3.3.4. Not Used	4
3.3.5. InterfaceBasicProfile	4
3.3.5.1. User Information	4

3.3.5.2. Security Information	4
3.3.5.3. User Preferences	4
3.3.6. Interface SecureProfile	4
3.3.6.1. get_authorization	4
3.3.6.2. get_authorization_classification	4
3.3.6.3. get_exception_by_countries	4
3.3.6.4. get_releasable_countries	4
3.3.6.5. get_releasable_organizations	4
3.3.6.6. get_distribution_limitation_codes	4
3.3.6.7. get_access_agreements	4
3.3.6.8. get_compartments	4
3.3.6.9. get_restricted_attributes	4
3.3.6.10. get_restricted_entities	4
3.3.7. Interface SecureViewProfile	4
3.3.7.1. use_authorization	4
3.3.7.2. get_restricted_attribute_values	4
3.3.8. Interface GIASProfile	4
3.3.8.1. get_allowable_operations	4
3.3.8.2. operation_is_allowed	4
3.3.8.3. list_volumes	4
3.3.8.4. new_folder	4
3.3.8.5. update_folder	4
3.3.8.6. remove_folder	4
3.3.8.7. list_folders	4
3.3.8.8. update_folder_item	4
3.3.8.9. get_entry	4
3.3.8.10. remove_entry	4
3.3.8.11. list_entries	4
3.3.8.12. list_entry_items	4
3.3.8.13. new_submitted_query	4
3.3.8.13. new_submitted_query_and_hitcount	4
3.3.8.14. new_submitted_hitcount	4
3.3.8.15. new_submitted_order	4
3.3.8.16. new_submitted_create	4
3.3.8.17. new_standing_query	4
3.3.8.18. new_standing_order	4
3.3.8.19. new_saved_query	4
3.3.8.20. update_saved_query	4
3.3.8.21. new_saved_order	4
3.3.8.22. update_saved_order	4
3.3.8.23. new_results_digest	4
3.3.8.24. update_results_digest	4
3.3.8.25. get_results_digest	4
3.3.8.26. new_saved_session	4
3.3.8.27. update_saved_session	4
3.3.8.28. get_saved_session	4
3.3.9. Data Types	4
3.3.9.1. SubmittedQuery and SubmittedQueryList	4
3.3.9.1. SubmittedQueryAndHitCount and SubmittedQueryAndHitCountList	4
3.3.9.2. SubmittedOrder and SubmittedOrderList	4
3.3.9.3. SubmittedCreate and SubmittedCreateList	4
3.3.9.4. SubmittedHitCount and SubmittedHitCountList	4
3.3.9.5. StandingQuery and StandingQueryList	4

3.3.9.6. StandingOrder and StandingOrderList.....	4
3.3.9.7. SavedQuery and SavedQueryList	4
3.3.9.8. SavedOrder and SavedOrderList	4
3.3.9.9. ResultsDigest and ResultsDigestList	4
3.3.9.10. SavedSession and SavedSessionList.....	4
3.4. PS EXCEPTIONS.....	4
3.4.1. UnknownProfileElementType.....	4
3.4.2. BadProfileElement.....	4
3.4.3. BadAccessCriteria.....	4
3.4.4. BadAccessValue.....	4
3.4.5. PasswordExpired.....	4
3.4.6. InvalidOrder	4
3.4.7. InvalidQuery.....	4
3.4.8. UnknownOperation.....	4
3.4.9. VolumeReadAccessDenied.....	4
3.4.10. VolumeWriteAccessDenied.....	4
3.4.11. FolderNotEmpty.....	4
3.4.12. FolderExists.....	4
3.4.13. UnknownFolderItemType	4
3.4.14. UnknownVolume	4
3.4.15. UnknownPRID	4
3.4.16. PermissionDenied.....	4
3.4.17. BadFileLocation.....	4
3.4.18. SystemFault	4
5. UNIVERSAL PRODUCT IDENTIFIER.....	4
5.1. UID	4
APPENDIX A: UCO IDL.....	4
APPENDIX D: UNIVERSAL PRODUCT IDENTIFICATION.....	4
APPENDIX E: ENCODING RULES FOR STRINGDAG.....	4
APPENDIX F: REFERENCE OMG STANDARD IDL.....	4
APPENDIX G: ACRONYMS.....	4
APPENDIX H: POINTS OF CONTACT.....	4

1. Overview

The USIGS Common Object Specification (UCOS) is a critical element to support interoperability in the USIGS architecture. The purpose of the UCOS is to define, in a single place, the interfaces, data types and error conditions that must be shared by multiple specifications in USIGS. By defining these shared elements in a single place, redundant (possibly non-interoperable) re-definitions of the same concept in multiple specifications are prevented. It is necessary for the effective use of this specification, that the elements defined here be used whenever and wherever appropriate and that all new shared elements identified by specifications be incorporated in later versions of this specification.

2. UCO Elements

2.1. Module UCO

All UCOS element definitions are enclosed within the UCO module.

```
module UCO
{
    ... all UCO elements...
};
```

2.2. UCO Data Types

2.2.1. UCO General Data Types

This section describes UCO general and simple composite geospatial data types.

2.2.1.1. Name, NameName, NameNameList

```
typedef string Name;

struct NameName
{
    Name name1;
    Name name2;
};

typedef sequence <NameName> NameNameList;
```

These three data types are used collectively as a container for a set of name (identifier) pairs. The data structure *NameNameList* represents an unbounded ordered list of name (identifier) pairs (i.e., *NameName*) that are records containing two *Names*.

2.2.1.2. NameList, NameValue, NameValueList

```
typedef sequence < Name > NameList;

struct NameValue
{
```

```
Name  aname;
any  value;
} ;

typedef sequence < NameValue > NameValueList;
```

These two data types (*NameValue*, *NameValueList*) are used collectively as a container for a name and its associated value. The data structure *NameValueList* represents an unbounded ordered list of name-valuepair (i.e., *NameValue*) that are records containing a name and its associated value.

The *NameValue* (i.e., name-valuepair) structure is used to associate an identifier defined as a type string with a value defined as a type any. The type any is used as a container to hold a value of any system or user defined type.

The *NameValueList* is used as an unbounded ordered list of *NameValue* pairs. This structure does not define any explicit relationship amongst the elements of the list except their order in the sequence.

The *NameList* is used as an ordered unbounded list of identifiers. This structure does not define any explicit relationship amongst the elements of the list except their order in the sequence.

2.2.1.3. FileLocation, Protocol, DirectAccessLocation, FileLocationList, and FileSize (j/NPS)

```
struct FileLocation
{
    string user_name;
    string password;
    string host_name;
    string path_name;
    string file_name;
};

enum Protocol { HTTP, FTP, FILE };

struct DirectAccessLocation
{
    Protocol      access_protocol;
```

```
    FileLocation  file_location;  
};  
  
typedef sequence <FileLocation> FileLocationList;  
  
typedef double Filesize;
```

These structures are used to identify individual and collections of files and file locations in a file system.

The FileLocation structure is used to define an individual file or file location as well as provide access control information needed to access that file or location. The system on which the file or location resides is defined by supplying a host name or IP number, as a string, in *host_name*. The location of that file on the specified system is defined by supplying, as a string, an absolute path to the directory in which the file resides in *path_name*. The syntax for this string uses the UNIX path specification (i.e., “/”) as a delimiter and all paths begin with a delimiter indicating the root directory. The identifier for the specific file is identified by supplying, as a string, the file name in *file_name*. If the FileLocation structure is being used to specify a directory, the full path including the desired directory will be included in *path_name*, and *file_name* will be NULL. The information needed to perform access control for the specified file is in the elements *user_name* and *password*.

An identifier for a user authorized to access the requested file or location is supplied as a string in *user_name*. The password that corresponds with that user identifier is supplied as a string in *password*. The values for these access control elements for public (“anonymous”) access are implementation dependent.

The structure FileLocationList is used to represent an ordered unbounded collection of files or file locations. Each file or location in the collection is a complete FileLocation structure, (i.e., there are no assumptions about the files or locations or their access control information with respect to each other). This structure does not define any explicit relationship amongst the elements of the collection except their order in the sequence.

The structure FileSize is used to describe the size in bytes of a file or data set.

The enumeration Protocol and the structure DirectAccessLocation add the information to specify the means by which a file may be accessed.

2.2.1.4. Percentage, Angle, Ratio, DateRate, and UCOS_NULL;

```
typedef float Percentage;  
  
typedef double Angle;
```

```
struct Ratio {  
    double numerator;  
    double denominator;  
};  
  
typedef double DataRate;  
typedef string UCOS_NULL;
```

2.2.1.5. Date, Time, TimeDuration, and AbsTime

```
struct Date  
{  
    unsigned short year;  
    unsigned short month;  
    unsigned short day;  
};  
  
struct Time  
{  
    unsigned short hour;  
    unsigned short minute;  
    float second;  
};  
  
typedef Time Duration;  
  
struct AbsTime  
{  
    Date aDate;  
    Time aTime;  
};
```

These structures are used to represent points in and periods of time.

The Date structure is used to represent a specific day. The year of that day is specified as an unsigned short in *year*. This must be the full specification of the year, using the last two digits (i.e., "97" for "1997") is

UNCLASSIFIED

considered non-compliant. The month of this day is specified as an unsigned short in *month* with a value between 1 and 12, where the value 1 represents January and the value 12 represents December. The specific day is identified as an unsigned short in *day*. To represent a general calendar day of the year, (i.e., Independence Day is 4 July) the year value shall be 0 (zero).

The structure Time can be used to represent both absolute and relative time. In order to represent absolute time, this structure is used with the Date structure to establish a point of reference (See the AbsTime structure defined below). To represent relative time or a length of time, the number of hours in the period is supplied as an unsigned short in *hour*, the number of minutes supplied as an unsigned short in *minute* and the number of seconds supplied as a float in *second*.

The structure AbsTime is used to represent an absolute point in time. It is composed of a Date structure (see above) and a Time structure. The Time structure is considered to represent the time in Military time (i.e., 24 hour clock).

2.2.1.6. EmailAddress

```
typedef string EmailAddress;
```

This data type serves to hold a definition of a complete email address. The syntax for this string is of the form *user_name* “@” *host_name*, where *user_name* is an identifier for a user or account on system *host_name*. *host_name* is an identifier for the target system. It can be in the form of a name or IP address.

2.2.1.7. DAG, DAGList, NodeID, NodeType, Node, NodeList, Cardinality, Edge, and EdgeList

```
typedef unsigned long NodeID;
enum NodeType { ROOT_NODE, ENTITY_NODE, RECORD_NODE,
                 ATTRIBUTE_NODE };

struct Node
{
    NodeID id;
    NodeType node_type;
    Name attribute_name;
    any value;
};
```

```

enum Cardinality { ONE_TO_ONE, ONE_TO_MANY, MANY_TO_ONE,
MANY_TO_MANY, ONE_TO_ZERO_OR_MORE, ONE_TO_ONE_OR_MORE,
ONE_TO_ZERO_OR_ONE };

struct Edge
{
    NodeID start_node;
    NodeID end_node;
    string relationship_type;
};

typedef sequence < Node > NodeList;
typedef sequence < Edge > EdgeList;

struct DAG
{
    NodeList nodes;
    EdgeList edges;
};

typedef sequence < DAG > DAGList;

```

A DAG is essentially a directed acyclic graph. Each DAG contains two types of information: data elements (“nodes”) and relationships among these elements (“edges”). The nodes are contained in the sequence *NodeList* and the edges are contained in the sequence *EdgeList*.

Each node contains an attribute-value pair. The name of the attribute is contained in the string *attribute_name* and the value is contained in the type any *value*. Along with this attribute-value pair is a NodeID (a type unsigned long) which uniquely identifies this node in this DAG, and a component which specifies the type of the Node, *node_type* (type *NodeType*).

The data structure represented in the IDL above is defined and composed of the following:

- *DAGList*: represents an ordered unbounded list of *DAG* elements.
- *DAG*: represents a directed acyclic graph, containing *NodeList* and *EdgeList* elements and can be a member of a *DAGList*.
- *NodeList*: represents a list of *Node* elements for a specific *DAG*.

- *EdgeList*: represents a list of *Edge* elements for a specific *DAG*.
- *Node*: is a 4-tuple containing node identification (i.e., *id*), node type (i.e., *node_type*), node attribute information (i.e., *attribute_name*), and an associated value (i.e., *value*).
- *Edge*: is a triplet defining the relationship (i.e., *relationship_type*) between two nodes.

2.2.1.8. StringDAG, StringDAGList, StringNode, StringNodeList

```

struct StringNode
{
    NodeID id;
    NodeType node_type;
    Name attribute_name;
    string value;
};

typedef sequence < StringNode > StringNodeList;

struct StringDAG
{
    UID::Product any prod;
    StringNodeList nodes;
    EdgeList edges;
};

typedef sequence < StringDAG > StringDAGList;

```

The structure StringDAG has essentially the same uses and form as the DAG defined above with two exceptions: 1) the StringDAG uses a type *string* rather than a type *any* as the container that holds the contents of each node 2) the StringDAG explicitly contains a [CORBA any \(prod\) which always contains a UID::Product](#). [This any which](#) identifies the Product that the DAG describes. The encoding rules which define how various data types are expressed in the string *value* of the Node are contained in appendix <X>.

2.2.1.9. ChangeType, Change, ChangeList, UpdateDAG and UpdateDAGList

```
enum ChangeType { ADD_CHANGE, UPDATE_CHANGE, DELETE_CHANGE } ;
struct Change { NodeID changed_node;
    ChangeType change_type; };
typedef sequence <Change> ChangeList;
struct UpdateDAG {
    DAG data;
    ChangeList changes;
} ;
typedef sequence <UpdateDAG> UpdateDAGList;
```

The UpdateDAG and UpdateDAGList are extensions of the DAG and DAGList structures defined above to add the ability to define changes to a DAG. This is done by combining a DAG with a ChangeList. A ChangeList indicates which nodes in the DAG have changed and what form of change has taken place (add, update or delete).

2.2.1.10. RequestDescription and RequestDescriptionList (j/NPS)

NOTE: These structures have temporarily been placed in the UCO specification. They will be moved to another specification in a later version of UCO.

```
struct RequestDescription
{
    string user_info;
    string request_type;
    string request_info;
    NameValueList request_details;
} ;
typedef sequence < RequestDescription > RequestDescriptionList;
```

The structure *RequestDescription* is used to describe the type and details of a request submitted for processing.

The *RequestDescription* structure is composed of four elements. The string *user_info* contains a message supplied by the submitting client, the contents of this message are completely determined by the client. The string *request_type* identifies the operation that was used to submit the request. The values and syntax of this element are defined in the appropriate GIAS profile. The string *request_info* contains any message the processing server wishes to return to the client concerning the specific request. It is intended to be human readable and is implementation dependent. The *NameValueList request_details* is intended to describe the

UNCLASSIFIED

parameters of the operation that generated this request. The specific names and values in this element are dependent on the operation that initiated this request and will be defined in the appropriate GIAS profile.

The structure *RequestDescriptionList* is an ordered unbounded list of *RequestDescriptions*.

2.2.1.11. BinData

```
typedef sequence<octet> BinData;
```

This structure is a general purpose container for the transfer of binary data. The internal structure and interpretation of this data is determined by the context in which it is used.

2.2.1.12. XMLDocument

```
typedef string XMLDocument;
```

This data type contains a string that is encoded in eXtensible Markup Language (XML) as defined in *Extensible Markup Language (XML) 1.0* specification .

2.2.1.13. EntityGraph, EntityNode, EntityRelationship, EntityNodeList, and EntityRelationshipList

```
struct EntityNode
{
    NodeID id;
    string entity_name;
};

struct EntityRelationship
{
    NodeID start_node;
    NodeID end_node;
    Cardinality start_to_end_card;
    Cardinality end_to_start_card;
};

typedef sequence < EntityNode > EntityNodeList;
```

```
typedef sequence < EntityRelationship >
EntityRelationshipList;

struct EntityGraph
{
    EntityNodeList nodes;
    EntityRelationshipList relationship;
};
```

The EntityGraph structure is used to describe an Entity-Relationship model.

2.2.2. Vector-like Data Types

The following data types are used to represent geospatial and geometric elements.

N.B.: These data types may be replaced in the future with definitions from the Open GIS Consortium (OGC), a standards forum in the GIS community. The OGC data types were not available at the time of development of this specification. When those data types definitions do become available, they will be reviewed and considered for inclusion in this document to replace the data types defined in this section.

2.2.2.1. Coordinate2d and Coordinate3d

```
struct Coordinate2d {
    double x;
    double y;
};

struct Coordinate3d {
    double x;
    double y;
    double z;
};
```

The *Coordinate2d* and *Coordinate3d* data types serve as a basis for all composite geospatial data types in UCO. The intent of *Coordinate2d* and *Coordinate3d* are to represent two- and three-dimensional points in a coordinate system. Their usage, alone or within other structures containing the coordinate, not the coordinate structures themselves, define the definition of the coordinate system being used for a specific coordinate.

The values of the point to be represented are placed in the *x*, *y*, and *z* elements of the Coordinate type as appropriate for either a two- or three- dimensional point. The mapping of coordinate system dimension to its elements is coordinate system dependent.

2.2.2.2. LineString2d and LineString3d

```
typedef sequence < Coordinate2d > LineString2d;
typedef sequence < Coordinate3d > LineString3d;
```

The *LineString2d* and *LineString3d* data types are used to represent a piece-wise linear path through two-dimensional space and three-dimensional space, respectively. The path is constructed by connecting the individual coordinates with line segments in the order they appear in the sequence. The order of the coordinates in the sequence is used only to define the connections of the coordinates; it does not necessarily imply a direction for either the line string or a temporal sequence (trajectory).

2.2.2.3. Polygon and PolygonSet

```
typedef sequence < Coordinate2d > Polygon;
typedef sequence < Polygon > PolygonSet;
```

The *Polygon* data type is used to represent a closed plane figure having three or more sides in two-dimensional space. The *Polygon* data types are an extension of the *LineString* data types with the constraint that the first point and last point in the sequence will be connected to close the polygon.

The *Coordinate2d* points are defined to be in a clockwise order. A *PolygonSet* is defined as a collection of *Polygon* data types.

2.2.2.4. Rectangle, RectangleList, LinearDimension, AreaDimension, Height, Elevation, Distance, Radius, Angle, Circle, Ellipse, and Weight

```
struct LinearDimension
{
    double      dimension;
    string      reference_system;
};
```

```
typedef double AreaDimension;

typedef LinearDimension Height;
typedef LinearDimension Elevation;
typedef LinearDimension Distance;
typedef LinearDimension Radius;
typedef double Angle;

struct Rectangle
{
    Coordinate2d upper_left;
    Coordinate2d lower_right;
};

typedef sequence < Rectangle > RectangleList;

struct Circle
{
    Coordinate2d centerpoint;
    Radius         aRadius;
};

struct Ellipse
{
    Coordinate2d centerpoint;
    Distance      minor_axis_len;
    Distance      major_axis_len;
    Angle         north_angle;
};

typedef double Weight;
```

UNCLASSIFIED

The *Rectangle* data structure is intended to provide a simple definition of an area on a flat surface. It is defined by supplying two *Coordinate2d* data structures indicating the opposing corners of the rectangle of interest. By convention, for coordinate systems describing the surface of the Earth, the *upper_left* element will indicate the Northwest corner and the *lower_right* element will indicate the Southeast corner.

The *Circle* data structure defines a circle on a flat surface. It is defined by supplying two components, *radius* and *centerpoint*.

The *Ellipse* data structure defines an ellipse based on a flat surface. It is defined by supplying the following components: *centerpoint*, *minor_axis_len*, *major_axis_len*, and *north_angle* for the *Ellipse*.

The structure *LinearDimension* acts as a basic structure to quantify a one-dimensional spatial measurement. It is composed of two elements that together provide a complete description of the dimension. The element *dimension* indicates the numerical value of the measurement and the element *reference_system* provides any needed context for the interpretation of the measurement. The related type *Height* indicates the measurement is a linear dimension above a reference point. In this case the element *reference_system* indicates the nature or identifier for that reference point. The type *Elevation* is similar. For types *Distance*, *Angle*, *Weight* and *Radius*, the use of element *reference_system* is implementation dependent.

The data type *AreaDimension* acts to define a two-dimensional spatial measurement (i.e., an area).

2.2.3. Raster-like Data Types

2.2.3.1. BufferType and Buffer

```
enum BufferType
{
    OCTET_DATA, CHAR_DATA, SHORT_DATA, USHORT_DATA, LONG_DATA,
    ULONG_DATA, FLOAT_DATA, DOUBLE_DATA
};

typedef sequence < octet > octetList;
typedef sequence < char > charList;
typedef sequence < unsigned short > ushortList;
typedef sequence < short > shortList;
typedef sequence < unsigned long > ulongList;
typedef sequence < long > longList;
typedef sequence < float > floatList;
typedef sequence < double > doubleList;
```

```

union Buffer
{
    switch (BufferType)
    {
        case OCTET_DATA: octetList octet_buffer;
        case CHAR_DATA:   charList char_buffer;
        case USHORT_DATA: ushortList ushort_buffer;
        case SHORT_DATA:  shortList short_buffer;
        case ULONG_DATA:  ulongList ulong_buffer;
        case LONG_DATA:   longList long_buffer;
        case FLOAT_DATA:  floatList float_buffer;
        case DOUBLE_DATA: doubleList double_buffer;
    };
}

```

2.2.3.2. SimpleGSImage

```

struct SimpleGSImage
{
    unsigned long width;
    unsigned long height;
    Buffer pixels;
};

```

The SimpleGSImage is intended to provide a basic definition of a simple grayscale image. This simple image has two attributes, *width* and *height*, which define the layout of the pixel data contained in Buffer *pixels*. The Buffer which holds the pixel data is of length *width* * *height*. Each value in the Buffer indicates the value of an individual pixel, beginning at the upper left corner of the image and continuing across the top of the image for *width* pixels.

2.2.3.3. SimpleCImage and CompressedImage

```

struct SimpleCImage
{

```

UNCLASSIFIED

```

    unsigned long width;
    unsigned long height;
    Buffer red_pixels;
    Buffer green_pixels;
    Buffer blue_pixels;
} ;

```

The SimpleCImage is intended to provide a basic definition of a simple color image with three bands. This simple image has two attributes, *width* and *height*, which define the layout of the pixel data contained in the three Buffers. The nth color pixel of this image is defined by the triplet composed of the nth element of each of the three Buffers. Each Buffer is of length *width* * *height*. Each element of the Buffer indicates the value of a color component (red, green or blue) of an individual pixel, beginning at the upper left corner of the image and continuing across the top of the image for *width* pixels. The type of data in each of the three pixel Buffers will be the same.

```

struct CompressedImage
{
    unsigned long width;
    unsigned long height;
    string compression_form;
    octetList data;
} ;

```

The CompressedImage is intended to provide a basic definition of a simple compressed image. This compressed image has two attributes, *width* and *height*, which define the dimensions of the pixel data contained in the *data* element. The *data* element holds the pixel data as an opaque binary stream. The compression form type determines the internal structure of the pixel data. The *compression_form* element holds an identifier, which indicates the type of compression.

2.2.4. State and Status (j/NPS)

```

enum State
{
    COMPLETED, IN_PROGRESS, ABORTED, CANCELED, PENDING,
    SUSPENDED, RESULTS_AVAILABLE, TRANSFER_COMPLETE
} ;

```

```

struct Status
{
    State completion_state;
    boolean warning;
    string status_message;
};

```

The Status structure and State enumeration type are intended to represent the current condition of a process, request, or other system element.

The enumeration State defines 8 (eight) identifiers (i.e., “states”) for the condition of a process, request or system element. The identifiers and the corresponding condition are defined as follows:

State	Description of Conditions
COMPLETED	All requested processing has completed successfully
IN_PROGRESS	Still processing, no error or abnormal conditions yet encountered
ABORTED	Processing has stopped due to an error or abnormal condition
CANCELED	Processing has been stopped by request
PENDING	Processing has not yet begun or has temporarily been halted
SUSPENDED	Processing has been temporarily suspended by client request
RESULTS_AVAILABLE	Processing has generated some results and made them available. Additional processing may occur.
TRANSFER_COMPLETE	Data has been completely transferred.

The structure Status is used to describe the details of the current condition of a process, request, or system element. The Status structure is composed of three elements: *completion_state*, a State (see above) indicating the current condition of the process, request or system element; *warning*, a boolean that if TRUE indicates that the *status_message* field contains warning information; and *status_message*, a string containing a human readable message that further amplifies or clarifies the State.

2.3. UCO Interfaces

Presently there are no interfaces defined for UCOS.

2.4. UCO Exceptions (j/NPS)

This section defines a generic exception model that can be expanded for use by all USIGS specifications. The model defines three exceptions: InvalidInputParameter, ProcessingFault, and SystemFault. Each exception is used to express a category of error conditions. These categories are based on the following two aspects: 1) the condition that caused the error and 2) the information needed to recover from the error if possible.

All operations that use this exception model would define one of two "raises" clauses:

- 1) raises (InvalidInputParameter, ProcessingFault, SystemFault);
or for operations with no input parameters
- 2) raises (ProcessingFault, SystemFault);

The following sections define the exceptions and supporting types for this exception model.

2.4.1. Exception_details (j/NPS)

```
struct exception_details {  
    string exception_name;  
    boolean standard_exception_name;  
    string exception_desc;  
};
```

The structure *exception_details* contains information that describes the exception condition that occurred.

The string *exception_name* identifies the specific exception that occurred. This string will be one of the standardized exceptions defined in the IDL in the form of a string constant or an implementation specific string defined by the service implementer. The boolean, *standard_exception_name* will be TRUE if the *exception_name* string is a standard exception i.e., defined by an IDL string constant. If FALSE it indicates that the *exception_name* is implementation specific. The string *exception_desc* is a human readable description of the error condition.

2.4.2. InvalidInputParameter (j/NPS)

```
exception InvalidInputParameter {
    exception_details details;
    UCO::NameList badInputParameters;
};
```

InvalidInputParameter is used to indicate that a value supplied by the client is outside the acceptable range for that parameter. The information needed to recover from this error is the name of the specific parameter which caused the exception. This exception provides a completed exception_details structure and a NameList that identifies the specific parameter(s) the client supplied that was determined to be outside of the acceptable range(s). This exception is intended to express error conditions that are identifiable solely based on the input parameter(s).

An example of this type of exception would be: client supplied an integer parameter *size* of 128, when the acceptable range is 0-64. The information returned by the exception might look like:

```
exception_details {
    string exception_name; = "BadSize"
    boolean standard_exception_name; = FALSE
    string exception_desc; = "Size parameter is out of range"
};
```

2.4.3. ProcessingFault (j/NPS)

```
exception ProcessingFault {
    exception_details details;
};
```

ProcessingFault is used to indicate that although the client-supplied input was acceptable, the request could not be completed because of an error that occurred during processing of the client's request. The information needed to recover from this error is the specific condition that prevented successful completion of this request. This exception provides a completed exception_details structure.

An example of this type of exception would be: client correctly invoked an operation which required access to a specific data set, which was locked by another user. The information returned by the exception might look like:

UNCLASSIFIED

```
exception_details {  
    string exception_name; = "DataLocked"  
    boolean standard_exception_name; = FALSE  
    string exception_desc; = "Data Set is currently locked by another  
    user"  
};
```

2.4.4. SystemFault (j/NPS)

```
exception SystemFault {  
    exception_details details;  
};
```

SystemFault is used to indicate that an error occurred that is internal to the server e.g., not necessarily directly related to any specific aspect of the client's request. No additional information is needed for recovery since error was not due to client's action. This exception provides a completed exception_details structure.

An example of this type of exception would be: client correctly invoked an operation, however, a server component needed to process this request was temporarily off-line. The information returned by the exception may look like the following example:

```
exception_details {  
    string exception_name; = "ResourceUnavailable"  
    boolean standard_exception_name; = FALSE  
    string exception_desc; = "The CD-ROM Writer is currently  
    unavailable for use"  
};
```

There is also a standardized set of general purpose exceptions defined by industry which address the most common reasons for failures, including communication and network errors (See Appendix E).

3. Profile Service Elements

3.1. Module PS

All Profile Service (PS) element definitions are enclosed within the PS Module. The PS Module has a dependency to the UCO Module and its description is included in this specification.

```
Module PS
{
... all PS elements ...
};
```

3.2. PS Data Types

The PS data types define the necessary data structures needed by a client application to utilize the defined methods of the PS interface. A profile is composed of ProfileElements, which are each of a distinct type.

3.2.1. ProfileElementType, ProfileElementTypeList

```
typedef string ProfileElementType;
typedef sequence < ProfileElementType > ProfileElementTypeList;
```

Each ProfileElement has a specific type (ProfileElementType) that identifies the category of user characteristics that it describes. The structure ProfileElementTypeList is used to assemble a set of these ProfileElementTypes into a single list.

3.3. PS Interfaces

The PS module (i.e., *module PS*) defines three interfaces, *Profile*, *ProfileElement*, and *ProfileMgr*. The *Profile* and *ProfileMgr* interfaces provide a client with the capability to obtain a user profile and access and modify the data defined within that profile. The client contacts the *ProfileMgr* to gain access to the client's specific *Profile* using the operation *get_profile*. Having acquired access to the *Profile* the client can then get and set specific values within that *Profile* using the defined operations (*get_profile_element_types*, *get_profile_element* and *set_profile_element*).

In addition, the client can obtain the last user profile modification time and date by invoking the selector operation *get_last_update_time*. These operations are described in the following subsections.

3.3.1. Interface Profile

UNCLASSIFIED

```
interface Profile
{
    ProfileElementTypeList get_profile_element_types()
        raises (UCO::ProcessingFault, UCO::SystemFault);

        ProfileElement get_profile_element(in ProfileElementType
element_type)
            raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

        SecureProfile get_secure_profile_element
            (in UCO::NameValueList
trusted_access_criteria)
            raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

        SecureViewProfile get_secure_view_profile_element
            (in UCO::NameValueList
trusted_access_criteria,
            in GIAS::ViewName view)
            raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

        UCO::AbsTime get_last_update_time()
        raises (UCO::ProcessingFault, UCO::SystemFault);

        void get_profiled_views(out UCO::NameList view_list)
        raises (UCO::ProcessingFault, UCO::SystemFault);
};
```

3.3.1.1. **get_profile_element_types**

```
ProfileElementTypeList get_profile_element_types ()
```

UNCLASSIFIED

```
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This selector operation returns a list of type *ProfileElementTypeList*, which contains the profile element types defined within this Profile.

3.3.1.2. get_profile_element

```
ProfileElement get_profile_element
    (in ProfileElementType element_type)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This selector operation returns a *ProfileElement* for the requested *element_type*. A UCO::InvalidInputParameter exception, which returns the string value “UnknownProfileElementType” in the *exception_name* field of the structure *exception_details*, will be raised if the requested *element_type* is unknown.

3.3.1.3. get_secure_profile_element

```
SecureProfile get_secure_profile_element
    (in UCO::NameValueList
trusted_access_criteria)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation returns the secure profile element that contains user security information that does not change across views. The trusted access criteria limits the availability of this information. If the access criteria does not contain expected names, this operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessCriteria” in the *exception_name* field of the structure *exception_details*. If the access criteria does not contain expected values, the operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessValue” in the *exception_name* field of the structure *exception_details*.

3.3.1.4. get_secure_view_profile_element

```
SecureViewProfile get_secure_view_profile_element
    (in UCO::NameValueList
trusted_access_criteria,
```

```

                in GIAS::ViewName      view)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation returns the secure profile element that contains user security information that is view specific. The trusted access criteria limits the availability of this information. If the access criteria does not contain expected names, the operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessCriteria” in the *exception_name* field of the structure *exception_details*. If the access criteria does not contain expected values, the operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessValue” in the *exception_name* field of the structure *exception_details*.

3.3.1.5. get_profiled_views

```

void get_profiled_views(out UCO::NameList view_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

```

This operation returns a list of views that are valid for a particular profile.

3.3.1.6. get_last_update_time

```

UCO::AbsTime get_last_update_time()
raises (UCO::ProcessingFault, UCO::SystemFault);

```

This selector operation returns the time the Profile was last modified.

3.3.2. Interface ProfileMgr

```

interface ProfileMgr
{
    Profile get_profile(in UCO::NameValueList access_criteria)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

    string get_version()
raises (UCO::ProcessingFault, UCO::SystemFault);

```

UNCLASSIFIED

```

void set_user_password
    (in UCO::NameValueList access_criteria,
     in string new_password)
    raises (UCO::InvalidInputParameter,
UCO::ProcessingFault, UCO::SystemFault);
};

```

3.3.2.1. get_profile

```

Profile get_profile (in UCO::NameValueList access_criteria)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation allows a client to gain access to its own Profile object. The access is gained by invoking this operation with the correct set of access criteria submitted as a NameValueList. The specific types and values for the access criteria are implementation dependent. If the access criteria does not contain expected names, this operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessCriteria” in the *exception_name* field of the structure *exception_details*. If the access criteria does not contain expected values, this operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessValue” in the *exception_name* field of the structure *exception_details*.

3.3.2.2. get_version

```

string get_version()
    raises (UCO::ProcessingFault, UCO::SystemFault);

```

This operation returns a string containing the version that this specific implementation of ProfileMgr is using.

3.3.2.3. set_user_password

```

void set_user_password
    (in UCO::NameValueList access_criteria,
     in string new_password)
    raises (UCO::InvalidInputParameter,
UCO::ProcessingFault, UCO::SystemFault);

```

This operation allows a client to set a new password. If the access criteria does not contain expected names, this operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessCriteria” in the *exception_name* field of the structure *exception_details*. If the access criteria does not contain expected values, this operation will raise a UCO::InvalidInputParameter exception that returns the string value “BadAccessValue” in the *exception_name* field of the structure *exception_details*.

3.3.3. Interface ProfileElement

```
interface ProfileElement
{
    UCO::AbsTime get_last_update_time()
        raises (UCO::ProcessingFault, UCO::SystemFault);
}
```

The ProfileElement object serves as the base abstract class for all types of content objects in a Profile. It contains operations common to all types of ProfileElement objects.

3.3.3.1. get_last_update_time

```
UCO::AbsTime get_last_update_time()
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the time this ProfileElement was last changed.

3.3.4. Not Used

3.3.5. Interface BasicProfile

The BasicProfile object is a specialization of the ProfileElement object to support the most general types of profile information. It contains operations and types common to all types of users.

```
interface BasicProfile : ProfileElement
{
    struct TelephoneNumber
    {
```

```
        string name;
        string number;
    } ;

typedef sequence<TelephoneNumber> TelephoneNumberList;

struct UserInformation
{
    string             name;
    string             organization;
    string             address;
    string             city;
    string             state;
    string             zip;
    string             country;
    string             email;
    TelephoneNumberList phone_numbers;
    UCO::FileLocation  ftp_location;
    UCO::AbsTime       password_expiration;
};

struct UserPreference
{
    string  name;
    string  value;
    boolean editable;
    string  description;
};

typedef sequence<UserPreference> UserPreferenceList;
typedef sequence<string> PreferenceNameList;

struct UserPreferenceDomain
```

UNCLASSIFIED

```
{  
    GIAS::Domain domain;  
    boolean      multi_select;  
};  
  
struct SecurityInformation  
{  
    string  classification;  
    boolean security_administrator_flag;  
};  
  
void get_user_information      (out UserInformation      info)  
    raises (UCO::ProcessingFault, UCO::SystemFault);  
  
void set_user_information      (in  UserInformation      info)  
    raises ( UCO::InvalidInputParameter,  
UCO::ProcessingFault, UCO::SystemFault);  
  
void get_security_information  (out SecurityInformation info)  
    raises (UCO::ProcessingFault, UCO::SystemFault);  
  
void get_available_preferences (out PreferenceNameList  
names)  
    raises (UCO::ProcessingFault, UCO::SystemFault);  
  
void get_preference_domain     (in  string preference_name,  
                                out GIAS::Domain      domain)  
    raises ( UCO::InvalidInputParameter,  
UCO::ProcessingFault, UCO::SystemFault);  
  
void get_user_preference       (in  string  preference_name,  
                                out UserPreference  
preference)
```

UNCLASSIFIED

```
        raises ( UCO::InvalidInputParameter,
UCO::ProcessingFault, UCO::SystemFault);

    void get_user_preferences      (out UserPreferenceList list)
        raises (UCO::ProcessingFault, UCO::SystemFault);

    void set_user_preference      (in UserPreference
preference)
        raises ( UCO::InvalidInputParameter,
UCO::ProcessingFault, UCO::SystemFault);

    void set_user_preferences     (in UserPreferenceList
preferences)
        raises ( UCO::InvalidInputParameter,
UCO::ProcessingFault, UCO::SystemFault);

};

}
```

3.3.5.1. User Information

```
void get_user_information(out UserInformation info)
    raises (UCO::ProcessingFault, UCO::SystemFault);
void set_user_information(in UserInformation info)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

These operations allow a client to get and set the current values of their user information. This information is expressed as a completed UserInformation structure.

3.3.5.2. Security Information

```
void get_security_information(out SecurityInformation info)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the current values of the security information.

3.3.5.3. User Preferences

```
void get_available_preferences (out PreferenceNameList names)
```

UNCLASSIFIED

```
    raises (UCO::ProcessingFault, UCO::SystemFault);  
  
void get_preference_domain(in string preference_name,  
                           out GIAS:: Domain adomain)  
    raises ( UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
void get_user_preference(in string preference_name,  
                        out UserPreference preference)  
    raises ( UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
void get_user_preferences (out UserPreferenceList list)  
    raises (UCO::ProcessingFault, UCO::SystemFault);  
  
void set_user_preference (in UserPreference    preference)  
    raises ( UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
void set_user_preferences (in UserPreferenceList preferences);  
    raises ( UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);
```

These operations allow a client to manage their preferences. The operation `get_available_preferences` returns a list of all preferences known or acceptable to the `ProfileElement`. The operation `get_preference_domain` returns a `Domain` type that indicates what form the preference is expressed in. The operations `get_user_preference` and `get_user_preferences` allow a client to retrieve the value of one or all of their current set preferences. Similarly, `set_user_preference` and `set_user_preferences` allow a client to set the current value(s) of their preferences.

3.3.6. Interface SecureProfile

The `SecureProfile` object is a specialization of the `ProfileElement` object to support basic security information access.

```
interface SecureProfile : ProfileElement  
{
```

UNCLASSIFIED

```
struct Authorization
{
    string          authorization_classification;
    UCO::NameList  exception_country_list;
    UCO::NameList  releasable_country_list;
    UCO::NameList  releasable_org_list;
    UCO::NameList  distribution_limit_code_list;
    UCO::NameList  access_agreement_list;
    UCO::NameList  compartment_list;
};

void      get_authorization
           (out Authorization      aauthorization)
raises (UCO::ProcessingFault, UCO::SystemFault);

void      get_authorization_classification
           (out string            classification)
raises (UCO::ProcessingFault, UCO::SystemFault);

void      get_exception_by_countries
           (out UCO::NameList     country_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

void      get_releasable_countries
           (out UCO::NameList     country_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

void      get_releasable_organizations
           (out UCO::NameList     organization_list)
raises (UCO::ProcessingFault, UCO::SystemFault);
```

UNCLASSIFIED

```
void      get_distribution_limitation_codes
          (out UCO::NameList      code_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

void      get_access_agreements
          (out UCO::NameList      agreement_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

void      get_compartments
          (out UCO::NameList      compartment_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

enum   AccessType {READ_DENIED, WRITE_DENIED};

void    get_restricted_attributes
          (in  AccessType     access_type,
           out UCO::NameList attribute_list)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void    get_restricted_entities
          (in  AccessType     access_type,
           out UCO::NameList entity_list)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
};
```

3.3.6.1. get_authorization

```
void    get_authorization (out Authorization      aauthorization)
raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the authorization information.

3.3.6.2. get_authorization_classification

```
void    get_authorization_classification
UNCLASSIFIED
```

```
        (out string          classification)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns a user's authorization classification level.

3.3.6.3. get_exception_by_countries

```
void      get_exception_by_countries
          (out UCO::NameList      country_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the exception by country list.

3.3.6.4. get_releasable_countries

```
void      get_releasable_countries
          (out UCO::NameList      country_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the releasable country list.

3.3.6.5. get_releasable_organizations

```
void      get_releasable_organizations
          (out UCO::NameList      organization_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the releasable organization list.

3.3.6.6. get_distribution_limitation_codes

```
void      get_distribution_limitation_codes
          (out UCO::NameList      code_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the distribution limitation code list.

3.3.6.7. get_access_agreements

```
void      get_access_agreements
```

```
(out UCO::NameList      agreement_list)
  raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the access agreement list.

3.3.6.8. get_compartments

```
void    get_compartments
        (out UCO::NameList      compartment_list)
  raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns the compartment list for the user.

3.3.6.9. get_restricted_attributes

```
void    get_restricted_attributes
        (in  AccessType    access_type,
         out UCO::NameList attribute_list)
  raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
          UCO::SystemFault);
```

This operation returns the restricted attribute identifiers for the user relative to a specific view.

3.3.6.10. get_restricted_entities

```
void    get_restricted_entities
        (in  AccessType    access_type,
         out UCO::NameList entity_list)
  raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
          UCO::SystemFault);
```

This operation returns the restricted entity identifiers for the user relative to a specific view.

3.3.7. Interface SecureViewProfile

The SecureViewProfile object is a specialization of the SecureProfile object to support view-oriented security information access.

```
interface SecureViewProfile : SecureProfile
{
```

UNCLASSIFIED

```

boolean use_authorization()
    raises (UCO::ProcessingFault, UCO::SystemFault);

void get_restricted_attribute_values
    (out string restriction)
    raises (UCO::ProcessingFault, UCO::SystemFault);
} ;

```

3.3.7.1. use_authorization

```

boolean use_authorization()
    raises (UCO::ProcessingFault, UCO::SystemFault);

```

This operation returns a true indication if the authorization is to be used for discretionary access control for the specified view.

3.3.7.2. get_restricted_attribute_values

```

void get_restricted_attribute_values (out string
restriction)
    raises (UCO::ProcessingFault, UCO::SystemFault);

```

This operation returns the restricted attribute value string for the user relative to a specific view.

3.3.8. Interface GIASProfile

The GIASProfile object is a specialization of the ProfileElement object to support users of GIAS Libraries. It contains operations and types specific to GIAS libraries' operations.

```

interface GIASProfile : ProfileElement
{
    /**
     * Defines a profile id used to uniquely identify objects
     * within the

```

```
// user profile
// ***
typedef string PRID;
typedef sequence <PRID> PRIDLList;

void get_allowable_operations(out UCO::NameList
operation_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);

boolean operation_is_allowed (in string operation)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

typedef string FolderTree;
// string defined as: \folder1\folder2\folder3\etc...
// An empty string or '\' denotes the root folder.

struct FolderLocation
{
    string volume;
    FolderTree folder;
};

struct VolumeInfo
{
    boolean default_volume;
    string volume;
    boolean read_permission;
    boolean write_permission;
    boolean create_delete_permission;
};
```

```
typedef      sequence < VolumeInfo > VolumeInfoList;
typedef      sequence < FolderLocation> FolderLocationList;

enum FolderItemType
{
    SAVED_QUERY,
    SUBMITTED_QUERY,
    SUBMITTED_QUERY_AND_HIT_COUNT,

    STANDING_QUERY,
    SAVED_ORDER,
    SUBMITTED_ORDER,
    STANDING_ORDER,
    SUBMITTED_CREATE,
    SUBMITTED_HIT_COUNT,
    RESULTS_DIGEST,
    SESSION,
    ALL
};

struct FolderItem
{
    PRID           item_id;
    FolderLocation location;
    string          name;
    string          description;
    FolderItemType type;
    UCO::AbsTime   creation_time;
    UCO::AbsTime   last_accessed_time;
    UCO::AbsTime   last_modified_time;
    string          owner_name;
    string          user_created_name;
}
```

UNCLASSIFIED

```
        string          user_last_accessed_name;
        string          user_last_modified_name;
        UCO::Rectangle  area_of_interest_mbr;
    };

typedef sequence < FolderItem >    FolderItemList;

// ***
// FolderItem holds one of the following:
//   SubmittedQuery
//   SubmittedQueryAndHitCount
//   SubmittedOrder
//   SubmittedCreate
//   StandingQuery
//   StandingOrder
//   SavedQuery
//   SavedOrder
//   ResultsDigest
//   SavedSession
// ***
typedef any FolderContent;
typedef sequence <FolderContent> FolderContentList;

enum SearchDepth
{
    SINGLE_FOLDER,
    FOLDER_TREE
};

void list_volumes(
    out VolumeInfoList volume_list)
raises (UCO::ProcessingFault, UCO::SystemFault);
UNCLASSIFIED
```

```
void new_folder (
    in FolderLocation folder)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void update_folder (
    in FolderLocation folder,
    in FolderLocation new_location)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void remove_folder (
    in FolderLocation folder)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void list_folders (
    in FolderLocation      folder,
    out FolderLocationList query_items)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void update_folder_item (
    in PRID            item_id,
    in FolderLocation   location,
    in string          name,
    in string          description,
    in UCO::Rectangle  area_of_interest_mbr)
```

UNCLASSIFIED

```
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void get_entry (
    in PRID             item_id,
    out FolderContent   entry)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void remove_entry (
    in PRID             item_id)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void list_entries (
    in FolderItemType   type,
    in FolderLocation   starting_point,
    in SearchDepth      depth,
    out FolderContentList entries)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void list_entry_items (
    in FolderItemType   type,
    in FolderLocation   starting_point,
    in SearchDepth      depth,
    out FolderItemList  entry_items)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

```
struct SubmittedQuery
{
    FolderItem           item_info;
    GIAS::SubmitQueryRequest request;
};

typedef sequence<SubmittedQuery> SubmittedQueryList;

struct SubmittedQueryAndHitCount
{
    FolderItem           item_info;
    GIAS::SubmitQueryRequest request;
    GIAS::RequestList    hit_count_requests;
};

typedef sequence<SubmittedQueryAndHitCount>
SubmittedQueryAndHitCountList;

PRID new_submitted_query (
    in FolderLocation          folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle           area_of_interest_mbr,
    in GIAS::SubmitQueryRequest request)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

PRID new_submitted_query_and_hitcount (
    in FolderLocation          folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle           area_of_interest_mbr,
```

UNCLASSIFIED

```
    in GIAS::SubmitQueryRequest    request,
    in GIAS::RequestList          hit_count_requests)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

PRID new_submitted_hitcount (
    in FolderLocation           folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle           area_of_interest_mbr,
    in GIAS::HitCountRequest   request)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct SubmittedOrder
{
    FolderItem                  item_info;
    GIAS::OrderRequest          request;
};

typedef sequence<SubmittedOrder> SubmittedOrderList;

PRID new_submitted_order(
    in FolderLocation           folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle           area_of_interest_mbr,
```

UNCLASSIFIED

```
        in GIAS::OrderRequest order)
        raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct SubmittedCreate
{
    FolderItem           item_info;
    GIAS::CreateRequest request;
};

typedef sequence <SubmittedCreate> SubmittedCreateList;

PRID new_submitted_create(
    in FolderLocation      folder,
    in string               name,
    in string               description,
    in UCO::Rectangle       area_of_interest_mbr,
    in GIAS::CreateRequest create_request)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct StandingQuery
{
    FolderItem           item_info;
    GIAS::SubmitStandingQueryRequest request;
};

typedef sequence <StandingQuery> StandingQueryList;

PRID new_standing_query (
```

```
    in FolderLocation           folder,
    in string                  name,
    in string                  description,
    in UCO::Rectangle          area_of_interest_mbr,
                               in GIAS::SubmitStandingQueryRequest request)
                               raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct StandingOrder
{
    FolderItem           item_info;
    GIAS::QueryOrderContents   order;
    GIAS::SubmitStandingQueryRequest query;
};

typedef sequence <StandingOrder> StandingOrderList;

PRID new_standing_order(
    in FolderLocation           folder,
    in string                  name,
    in string                  description,
    in GIAS::QueryOrderContents order,
    in UCO::Rectangle          area_of_interest_mbr,
                               in GIAS::SubmitStandingQueryRequest query)
                               raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

```
struct SavedQuery
{
    FolderItem           item_info;

    GIAS::Query          bqs;
    UCO::FileLocation    thumbnail_location;
    boolean               browse_image_returned_flag;
    UCO::NameList         result_attributes;
    GIAS::SortAttributeList sort_attributes;
    string                geographic_datum;
    UCO::AbsTime          last_submitted_date;
};

typedef sequence < SavedQuery >   SavedQueryList;

PRID new_saved_query (
    in FolderLocation           folder,
    in string                    name,
    in string                    description,
    in GIAS::Query               bqs,

    in boolean                  browse_image_returned_flag,
    in UCO::NameList             result_attributes,
    in GIAS::SortAttributeList   sort_attributes,
    in string                    geographic_datum,
    in UCO::AbsTime              last_submitted_date,
    in UCO::Rectangle            area_of_interest_mbr)

    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

```
void update_saved_query(
    in PRID             saved_query_id,
    in string            name,
    in string            description,
    in GIAS::Query      bqs,
in boolean          browse_image_returned_flag,
    in UCO::NameList     result_attributes,
    in GIAS::SortAttributeList sort_attributes,
    in string            geographic_datum,
    in UCO::AbsTime      last_submitted_date,
    in UCO::Rectangle    area_of_interest_mbr)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct SavedOrder
{
    FolderItem   item_info;
    GIAS::OrderContents   order;
};

typedef sequence < SavedOrder > SavedOrderList;

PRID new_saved_order (
    in FolderLocation   folder,
    in string            name,
    in string            description,
    in UCO::Rectangle    area_of_interest_mbr,
    in GIAS::OrderContents   order)
```

UNCLASSIFIED

```
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void update_saved_order(
    in PRID          saved_order_id,
    in string        name,
    in string        description,
    in UCO::Rectangle area_of_interest_mbr,
    in GIAS::OrderContents   order)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct ResultsDigest
{
    FolderItem      item_info;
};

typedef sequence < ResultsDigest >  ResultsDigestList;

PRID new_results_digest (
    in FolderLocation   folder,
    in string           name,
    in string           description,
    in UCO::Rectangle   area_of_interest_mbr,
    in UCO::FileLocation digest_location)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void update_results_digest()
```

```
        in PRID                  results_digest_id,
        in string                 name,
        in string                 description,
        in UCO::Rectangle         area_of_interest_mbr,
        in UCO::FileLocation     digest_location)

    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void get_results_digest (
    in PRID                  query_id,
    in UCO::FileLocation     destination_file)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

struct SavedSession
{
    FolderItem      item_info;
    string          session_text;
};

typedef sequence < SavedSession >  SavedSessionList;

PRID new_saved_session (
    in FolderLocation      folder,
    in string                name,
    in string                description,
    in UCO::Rectangle        area_of_interest_mbr,
    in UCO::FileLocation     session_location)
```

```
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void update_saved_session(
    in PRID                 saved_session_id,
    in string                name,
    in string                description,
    in UCO::Rectangle         area_of_interest_mbr,
    in UCO::FileLocation     session_location)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void get_saved_session (
    in PRID                 query_id,
    in UCO::FileLocation     destination_file)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

};
```

3.3.8.1. **get_allowable_operations**

```
void get_allowable_operations(out UCO::NameList operation_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns a list of the allowable operations for a particular user.

3.3.8.2. **operation_is_allowed**

```
boolean operation_is_allowed (in string operation)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation returns TRUE if the specified manager is accessible by the user, otherwise FALSE is returned.

3.3.8.3. list_volumes

```
void list_volumes(
    out VolumeInfoList volume_list)
raises (UCO::ProcessingFault, UCO::SystemFault);
```

This operation returns a list of available volumes.

3.3.8.4. new_folder

```
void new_folder (in FolderLocation folder)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

Creates a new folder in the location specified.

3.3.8.5. update_folder

```
void update_folder (
    in FolderLocation folder,
    in FolderLocation new_location)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation moves a folder's location.

3.3.8.6. remove_folder

```
void remove_folder (
    in FolderLocation folder)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation removes an empty folder.

3.3.8.7. list_folders

```
void list_folders (
    in FolderLocation      folder,
```

UNCLASSIFIED

```

        out FolderLocationList query_items)
        raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation returns a list of folders in a specific folder.

3.3.8.8. update_folder_item

```

void update_folder_item (
    in PRID           item_id,
    in FolderLocation location,
    in string          name,
    in string          description,
    in UCO::Rectangle area_of_interest_mbr)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation updates a specific folder item in the Profile.

3.3.8.9. get_entry

```

void get_entry (
    in PRID           item_id,
    out FolderContent entry)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation returns the contents of a folder item. Note that some folder item types may override this operation.

3.3.8.10. remove_entry

```

void remove_entry (
    in PRID           item_id)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation removes a folder item.

3.3.8.11. list_entries

```
void list_entries (
    in FolderItemType      type,
    in FolderLocation      starting_point,
    in SearchDepth         depth,
    out FolderContentList  entries)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation returns a list of folder contents of a given type. The search can cover just the given folder or recursively search the folder tree starting at the given folder location depending on the depth specified. Note that this call may not be useful for some folder item types.

3.3.8.12. list_entry_items

```
void list_entry_items (
    in FolderItemType      type,
    in FolderLocation      starting_point,
    in SearchDepth         depth,
    out FolderItemList     entry_items)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation returns a list of folder information for a given folder item type. The search can cover just the given folder or recursively search the folder tree starting at the given folder location depending on the depth specified.

3.3.8.13. new_submitted_query

```
PRID new_submitted_query (
    in FolderLocation          folder,
    in string                  name,
    in string                  description,
    in UCO::Rectangle          area_of_interest_mbr,
    in GIAS::SubmitQueryRequest request)
    raises (UCO::InvalidInputParameter,
UCO::ProcessingFault, UCO::SystemFault);
```

UNCLASSIFIED

This operation creates a new submitted query at the specified folder location.

3.3.8.13. new_submitted_query_and_hitcount

```
PRID new_submitted_query_and_hitcount (
    in FolderLocation           folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle           area_of_interest_mbr,
    in GIAS::SubmitQueryRequest request,
    in GIAS::RequestList        hit_count_requests)
    raises (UCO::InvalidInputParameter,
    UCO::ProcessingFault, UCO::SystemFault);
```

This operation creates a new submitted query and hitcount at the specified folder location. It includes a list of Requests which are the hit_count_requests that correspond to this submitted query. This list may be empty i.e of length zero.

3.3.8.14. new_submitted_hitcount

```
PRID new_submitted_hitcount (
    in FolderLocation           folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle           area_of_interest_mbr,
    in GIAS::HitCountRequest   request)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);
```

This operation creates a new submitted hitcount at the specified folder location.

3.3.8.15. new_submitted_order

```
PRID new_submitted_order(
    in FolderLocation           folder,
    in string                   name,
```

```

        in string          description,
        in UCO::Rectangle area_of_interest_mbr,
        in GIAS::OrderRequest order)

    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation creates a new submitted order at the specified folder location.

3.3.8.16. new_submitted_create

```

PRID new_submitted_create(
    in FolderLocation      folder,
    in string              name,
    in string              description,
    in UCO::Rectangle      area_of_interest_mbr,
    in GIAS::CreateRequest create_request)

    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation creates a new submitted “create” at the specified folder location.

3.3.8.17. new_standing_query

```

PRID new_standing_query (
    in FolderLocation      folder,
    in string              name,
    in string              description,
    in UCO::Rectangle      area_of_interest_mbr,
    in GIAS::SubmitStandingQueryRequest request)

    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation creates a new standing query at the specified folder location.

3.3.8.18. new_standing_order

```

PRID new_standing_order(
    in FolderLocation      folder,

```

UNCLASSIFIED

```

        in string                               name,
        in string                               description,
        in GIAS::QueryOrderContents
order,
        in UCO::Rectangle
area_of_interest_mbr,
        in GIAS::SubmitStandingQueryRequest query)
        raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation creates a new standing order at the specified folder location.

3.3.8.19. new_saved_query

```

PRID new_saved_query (
        in FolderLocation           folder,
        in string                   name,
        in string                   description
        in GIAS::Query             bqs,

        in boolean                  browse_image_returned_flag,
        in UCO::NameList            result_attributes,
        in GIAS::SortAttributeList sort_attributes,
        in string                   geographic_datum,
        in UCO::AbsTime             last_submitted_date,
        in UCO::Rectangle           area_of_interest_mbr)
        raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

```

This operation saves the query information to the specified folder.

3.3.8.20. update_saved_query

```

void update_saved_query(
        in PRID          saved_query_id,
        in string         name,

```

```
    in string          description,
    in GIAS::Query    bqs,

    in boolean         browse_image_returned_flag,
    in UCO::NameList  result_attributes,

    in GIAS::SortAttributeList  sort_attributes,
    in string          geographic_datum,
    in UCO::AbsTime    last_submitted_date,
    in UCO::Rectangle  area_of_interest_mbr)

raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault, UCO::SystemFault );
```

This operation updates a specified query from the Profile.

3.3.8.21. new_saved_order

```
PRID new_saved_order (
    in FolderLocation   folder,
    in string           name,
    in string           description,
    in UCO::Rectangle   area_of_interest_mbr,
    in GIAS::OrderContents  order)

raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);
```

This operation creates a new saved order at the specified folder location.

3.3.8.22. update_saved_order

```

void update_saved_order(
    in PRID             saved_order_id,
    in string            name,
    in string            description,
    in UCO::Rectangle    area_of_interest_mbr,
    in GIAS::OrderContents   order)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

```

This operation updates a saved order at the specified folder location.

3.3.8.23. new_results_digest

```

PRID new_results_digest (
    in FolderLocation    folder,
    in string            name,
    in string            description,
    in UCO::Rectangle    area_of_interest_mbr,
    in UCO::FileLocation digest_location)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

```

This operation saves the results digest to the specified folder. The server will retrieve the digest from the specified file location.

3.3.8.24. update_results_digest

```

void update_results_digest(
    in PRID             results_digest_id,
    in string            name,
    in string            description,
    in UCO::Rectangle    area_of_interest_mbr,
    in UCO::FileLocation digest_location)
raises (

```

UNCLASSIFIED

```
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation will update a results digest at the specified file location.

3.3.8.25. get_results_digest

```
void get_results_digest (
    in PRID                 query_id,
    in UCO::FileLocation destination_file)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation retrieves the results digest for the specified ID. The server will transfer the digest to the specified file location.

3.3.8.26. new_saved_session

```
PRID new_saved_session (
    in FolderLocation      folder,
    in string              name,
    in string              description,
    in UCO::Rectangle       area_of_interest_mbr,
    in UCO::FileLocation    session_location)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

This operation saves the session to the specified folder. The server will transfer the session from the specified file location.

3.3.8.27. update_saved_session

```
void update_saved_session(
    in PRID                saved_session_id,
    in string              name,
    in string              description,
    in UCO::Rectangle       area_of_interest_mbr,
    in UCO::FileLocation    session_lcoation)
```

```

        raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

```

This operation updates the specified saved session in the Profile.

3.3.8.28. get_saved_session

```

void get_saved_session (
    in PRID                 query_id,
    in UCO::FileLocation destination_file)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault, UCO::SystemFault );

```

This operation retrieves the saved session for the specified ID. The server will transfer the session to the specified file location.

3.3.9. Data Types

3.3.9.1. SubmittedQuery and SubmittedQueryList

```

struct SubmittedQuery
{
    FolderItem           item_info;
    GIAS::SubmitQueryRequest request;
};

typedef sequence <SubmittedQuery> SubmittedQueryList;

```

These are queries that have been submitted to a Library, but the client has not yet completed the retrieval of the query results.

3.3.9.1. SubmittedQueryAndHitCount and SubmittedQueryAndHitCountList

```

struct SubmittedQueryAndHitCount
{
    FolderItem           item_info;
    GIAS::SubmitQueryRequest request;
};

```

```

        GIAS::RequestList           hit_count_requests;
    } ;

typedef sequence <SubmittedQueryAndHitCount>
SubmittedQueryHitCountList;

```

These are queries and their associated hit counts that have been submitted to a Library, but the client has not yet completed the retrieval of the query results.

3.3.9.2. SubmittedOrder and SubmittedOrderList

```

struct SubmittedOrder
{
    FolderItem           item_info;
    GIAS::OrderRequest  request;
} ;

typedef sequence<SubmittedOrder> SubmittedOrderList;

```

These are orders that have been submitted to a Library, but the user has not yet deleted.

3.3.9.3. SubmittedCreate and SubmittedCreateList

```

struct SubmittedCreate
{
    FolderItem           item_info;
    GIAS::CreateRequest  request;
} ;

typedef sequence<SubmittedCreate> SubmittedCreateList;

```

These are create requests that have been submitted to a Library, but the user has not yet deleted.

3.3.9.4. SubmittedHitCount and SubmittedHitCountList

```

struct SubmittedHitCount
{

```

```

    FolderItem           item_info;
    GIAS::HitCountRequest request;
};

typedef sequence<SubmittedHitCount> SubmittedHitCountList;

```

These are hit count requests that have been submitted to a Library, but the user has not yet deleted.

3.3.9.5. StandingQuery and StandingQueryList

```

struct StandingQuery
{
    FolderItem           item_info;
    GIAS::SubmitStandingQueryRequest request;
};

typedef sequence <StandingQuery> StandingQueryList;

```

These are queries that get executed on a scheduled basis.

3.3.9.6. StandingOrder and StandingOrderList

```

struct StandingOrder
{
    FolderItem           item_info;
    GIAS::QueryOrderContents          order;
    GIAS::SubmitStandingQueryRequest query;
};

typedef sequence <StandingOrder> StandingOrderList;

```

These are orders that get executed on a scheduled basis.

3.3.9.7. SavedQuery and SavedQueryList

```
struct SavedQuery
```

UNCLASSIFIED

```
{  
    FolderItem          item_info;  
  
    GIAS::Query        bqs;  
    UCO::FileLocation  thumbnail_location;  
    boolean            browse_image_returned_flag;  
    UCO::NameList      result_attributes;  
  
    GIAS::SortAttributeList sort_attributes;  
    string              geographic_datum;  
    UCO::AbsTime       last_submitted_date;  
};  
  
typedef sequence < SavedQuery > SavedQueryList;
```

These are queries that are being saved by the Profile Service.

3.3.9.8. SavedOrder and SavedOrderList

```
struct SavedOrder  
{  
    FolderItem          item_info;  
    GIAS::OrderContents order;  
};  
typedef sequence < SavedOrder > SavedOrderlist;
```

These are orders that are being saved by the Profile Service.

3.3.9.9. ResultsDigest and ResultsDigestList

```
struct ResultsDigest  
{  
    FolderItem          item_info;
```

```
}
```

```
typedef sequence < ResultsDigest > ResultsDigestList;
```

These are the results of previous Catalog queries.

3.3.9.10. SavedSession and SavedSessionList

```
struct SavedSession
{
    FolderItem      item_info;
    string          session_text;
};
```

```
typedef sequence < SavedSession > SavedSessionList;
```

These are the descriptions of sessions.

3.4. PS Exceptions

This section lists the string constant exception names that have been defined for the *module PS* for interfaces *Profile*, *ProfileElement* and *ProfileMgr*.

```
const string UnknownProfileElementTypeConst = "UnknownProfile
ElementType";
const string BadAccessCriteriaConst = "BadAccessCriteria";
const string BadProfileElementConst = "BadProfileElement";
const string BadAccessValueConst = "BadAccessValue";
const string PasswordExpiredConst = "PasswordExpired";
const string InvalidOrderConst = "InvalidOrder";
const string InvalidQueryConst = "InvalidQuery";
const string UnknownOperationConst = "UnknownOperation";
const string VolumeReadAccessDeniedConst =
"VolumeReadAccessDenied";
const string VolumeWriteAccessDeniedConst =
"VolumeWriteAccessDenied";
const string FolderNotEmptyConst = "FolderNotEmpty";
```

UNCLASSIFIED

```
const string FolderExistsConst = "FolderExists";
const string UnknownFolderItemTypeConst = "UnknownFolderItemType";
const string UnknownVolumeConst = "UnknownVolume";
const string UnknownPRIDConst = "UnknownPRID";
const string PermissionDeniedConst = "PermissionDenied";
const string BadFileLocationConst = "BadFileLocation";
const string SystemFaultConst = "SystemFault";
```

3.4.1. UnknownProfileElementType

```
const string UnknownProfileElementTypeConst = "UnknownProfile
ElementType";
```

This exception indicates that a profile element type supplied by a client in the selector operation *get_profile_element* or modifier operation *set_profile_element* was unknown by the server. The *exception_details* structure will identify the unacceptable profile element criteria submitted.

3.4.2. BadProfileElement

```
const string BadProfileElementConst = "BadProfileElement";
```

This exception indicates that a profile value supplied by a client in the modifier operation was either the wrong type or out of range. The *exception_details* structure will list the unacceptable profile element criteria submitted.

3.4.3. BadAccessCriteria

```
const string BadAccessCriteriaConst = "BadAccessCriteria";
```

This exception indicates the client has supplied incomplete, invalid or otherwise unacceptable access criteria. The *exception_details* structure will list the unacceptable access criteria submitted.

3.4.4. BadAccessValue

```
const string BadAccessValueConst = "BadAccessValue";
```

This exception indicates that one or more values supplied for access criteria were missing, incorrect or otherwise unacceptable. The *exception_details* structure will list which specific access criteria element(s) submitted were unacceptable and if appropriate the acceptable values or range of values.

3.4.5. PasswordExpired

```
const string PasswordExpiredConst = "PasswordExpired";
```

This exception indicates that the client has supplied an unacceptable (expired) password to gain access to a given system.

3.4.6. InvalidOrder

```
const string InvalidOrderConst = "InvalidOrder";
```

The order that a client has submitted can not be fulfilled due to an error with the contents of the order.

3.4.7. InvalidQuery

```
const string InvalidQueryConst = "InvalidQuery";
```

The query submitted by the client can not be processed by the Library it has been submitted to.

3.4.8. UnknownOperation

```
const string UnknownOperationConst = "UnknownOperation";
```

The client has submitted an operation that can not be processed by the Library the client is interacting with.

3.4.9. VolumeReadAccessDenied

```
const string VolumeReadAccessDeniedConst =
"VolumeReadAccessDenied";
```

A client has been denied access to a particular storage area of the Library even for read access.

3.4.10. VolumeWriteAccessDenied

```
const string VolumeWriteAccessDeniedConst =
"VolumeWriteAccessDenied";
```

A client has been denied access to a particular storage area of the Library for write access.

3.4.11. FolderNotEmpty

```
const string FolderNotEmptyConst = "FolderNotEmpty";
```

A client has specified a folder that is not empty.

3.4.12. FolderExists

```
const string FolderExistsConst = "FolderExists";
```

3.4.13. UnknownFolderItemType

```
const string UnknownFolderItemTypeConst = "UnknownFolderItemType";
```

3.4.14. UnknownVolume

```
const string UnknownVolumeConst = "UnknownVolume";
```

3.4.15. UnknownPRID

```
const string UnknownPRIDConst = "UnknownPRID";
```

3.4.16. PermissionDenied

```
const string PermissionDeniedConst = "PermissionDenied";
```

The user of a client has specified an operation that is not specified in user's profile.

3.4.17. BadFileLocation

```
const string BadFileLocationConst = "BadFileLocation";
```

3.4.18. SystemFault

```
const string SystemFaultConst = "SystemFault";
```

This exception indicates that a system fault has occurred and descriptive info should be provided on the nature of the fault.

N0104-G

UNCLASSIFIED

6 August 2001

5. Universal Product Identifier

5.1. UID

```
module UID
{
    interface Product
    {
    } ;

    typedef sequence < Product > ProductList;
}
```

The *UID* module serves as a universal product identifier that can be used for USIGS products. The *Product* interface serves as a unique handle or identifier for a data set or product. For example, a USIGS product that resides in a Library System. No operations are defined on this interface, its sole purpose is to serve as a USIGS universal identifier. *ProductList* is a data structure that can be utilized by USIGS services as a container for returning a list of *Products*.

Since there are no operations defined on this interface, no exceptions are defined.

Appendix A: UCO IDL

```
// ****
/*
/* The USIGS Common Object Specification
*/
/*
/*
/*
Description: Defines fundamental data types and
interfaces to be used by other specifications to
support interoperation across independently designed
interfaces.

/*
/*
/*
/*
History:
/*
Date Author Comment
---- -----
/* 15 May 97 D. Lutz Initial release for review
/* 2 July 97 D. Lutz Released for TEM review
/* 11 July 97 D. Lutz Changes based on 2 July TEM
Comments
/* 16 Oct 97 D. Lutz Changes based on 7 Oct TEM
Comments
/* 14 Nov 97 D. Lutz Changes based on 4 Nov TEM
Comments
/* 17 Dec 97 D. Lutz Changes based on 9 Dec TEM
Comments
/* 13 Apr 98 J. Baldo Changes based on feedback from
distributions of UCOS v1.1 and
GIAS v3.1
/* 2 July 98 J. Baldo/D. Lutz Changes based on feedback
from 22-23 June 98 TEM
/* 29 Sept 98 J. Baldo/D. Lutz Changes based on feedback
from 22-23 September 1998 TEM
```

```
/*
   * from 4-5 August 1999 TEM -
Simplified the 2 and 3D Coordinates to be floats only.

/*
   * from 19 January 2000 TEM -
/* included a generic exception's model that GIXS,
/*GIAS Profile will use.

/* 7 March 2000      from March 2000 TEM - added new /*
   * structure that is somewhat similar /*
to NameValueTable called a /*
   * StringTable

/*
/* Notes
/*
-----  

/* 7 March 2000      Needed to include the UID Module
/*
*****
```

```
#include "uid.idl"

// The USIGS Common Objects
module UCO
{
    // Generic data types

    typedef string Name;
    typedef sequence < Name > NameList;

    struct NameName
    {
        Name name1;
        Name name2;
    };
}
```

```
typedef sequence <NameName> NameNameList;

struct NameValue
{
    Name fname;
    any value;
};

typedef sequence < NameValue > NameValueList;

typedef float Percentage;

typedef double Angle;

struct Ratio {
    double numerator;
    double denominator;
};

typedef double DataRate;

typedef string UCOS_NULL;

struct FileLocation
{
    string user_name;
    string password;
    string host_name;
    string path_name;
    string file_name;
};
```

UNCLASSIFIED

```
typedef sequence < FileLocation > FileLocationList;

enum Protocol { HTTP, FTP, FILE };

struct DirectAccessLocation
{
    Protocol      access_protocol;
    FileLocation  file_location;
};

struct Date
{
    unsigned short year;
    unsigned short month;
    unsigned short day;
};

struct Time
{
    unsigned short hour;
    unsigned short minute;
    float second;
};

typedef Time Duration;

struct AbsTime
{
    Date aDate;
    Time aTime;
};
```

```
typedef string EmailAddress;

// Begin DAG definition

typedef unsigned long NodeID;

enum NodeType { ROOT_NODE, ENTITY_NODE, RECORD_NODE,
                ATTRIBUTE_NODE };

struct Node
{
    NodeID id;
    NodeType node_type;
    Name attribute_name;
    any value;
};

enum Cardinality { ONE_TO_ONE, ONE_TO_MANY, MANY_TO_ONE,
                  MANY_TO_MANY, ONE_TO_ZERO_OR_MORE, ONE_TO_ONE_OR_MORE,
                  ONE_TO_ZERO_OR_ONE };

struct Edge
{
    NodeID start_node;
    NodeID end_node;
    string relationship_type;
};

typedef sequence < Node > NodeList;
typedef sequence < Edge > EdgeList;

struct DAG
```

```
{  
    NodeList nodes;  
    EdgeList edges;  
};  
  
typedef sequence < DAG > DAGList;  
  
// Begin StringDAG definition  
struct StringNode  
{  
    NodeID id;  
    NodeType node_type;  
    Name attribute_name;  
    string value;  
};  
typedef sequence < StringNode > StringNodeList;  
  
struct StringDAG  
{  
    UID::ProductAny prod;  
    StringNodeList nodes;  
    EdgeList edges;  
};  
  
typedef sequence < StringDAG > StringDAGList;  
  
enum ChangeType { ADD_CHANGE, UPDATE_CHANGE, DELETE_CHANGE };  
  
struct Change {
```

```
NodeID changed_node;
ChangeType change_type;

};

typedef sequence <Change> ChangeList;

struct UpdateDAG {
    DAG data;
    ChangeList changes;

};

typedef sequence <UpdateDAG> UpdateDAGList;

struct RequestDescription
{
    string user_info;
    string request_type;
    string request_info;
    NameValueList request_details;
};

typedef sequence < RequestDescription > RequestDescriptionList;
typedef sequence <octet> BinData;

typedef string XMLDocument;

// Basic Geospatial data types
```

```
// 3D and 2D floating point coordinate
struct Coordinate2d {
    double x;
    double y;
};

struct Coordinate3d {
    double x;
    double y;
    double z;
};

struct LinearDimension
{
    double dimension;
    string reference_system;
};

typedef double AreaDimension;

typedef LinearDimension Height;
typedef LinearDimension Elevation;
typedef LinearDimension Distance;
typedef LinearDimension Radius;
typedef sequence < Coordinate2d > LineString2d;
typedef sequence < Coordinate3d > LineString3d;
typedef sequence < Coordinate2d > Polygon;
typedef sequence < Polygon > PolygonSet;

struct Circle
{
```

UNCLASSIFIED

```
Coordinate2d centerpoint;
Radius         aRadius;
};

struct Ellipse
{
    Coordinate2d centerpoint;
    Distance     minor_axis_len;
    Distance     major_axis_len;
    Angle        north_angle;
};

struct Rectangle
{
    Coordinate2d upper_left;
    Coordinate2d lower_right;
};

typedef sequence < Rectangle > RectangleList;

typedef double FileSize;

typedef double Weight;

// Simple composite geospatial datatypes

enum BufferType
{
    OCTET_DATA, CHAR_DATA, SHORT_DATA, USHORT_DATA,
    LONG_DATA, ULONG_DATA, FLOAT_DATA, DOUBLE_DATA
}
```

```
};

typedef sequence < octet > octetList;
typedef sequence < char >charList;
typedef sequence < unsigned short >ushortList;
typedef sequence < short >shortList;
typedef sequence < unsigned long >ulongList;
typedef sequence < long >longList;
typedef sequence < float >floatList;
typedef sequence < double >doubleList;

union Buffer
{
    switch (BufferType)
    {
        case OCTET_DATA: octetList octet_buffer;
        case CHAR_DATA:   charList char_buffer;
        case USHORT_DATA: ushortList ushort_buffer;
        case SHORT_DATA:  shortList short_buffer;
        case ULONG_DATA:  ulongList ulong_buffer;
        case LONG_DATA:   longList long_buffer;
        case FLOAT_DATA:  floatList float_buffer;
        case DOUBLE_DATA: doubleList double_buffer;
    };
}

struct SimpleGSImage
{
    unsigned long width;
    unsigned long height;
    Buffer pixels;
};
```

```
struct SimpleCImage
{
    unsigned long width;
    unsigned long height;
    Buffer red_pixels;
    Buffer green_pixels;
    Buffer blue_pixels;
};

struct CompressedImage
{
    unsigned long width;
    unsigned long height;
    string compression_form;
    octetList data;
};

enum State
{
    COMPLETED, IN_PROGRESS, ABORTED, CANCELED, PENDING,
    SUSPENDED, RESULTS_AVAILABLE, TRANSFER_COMPLETE
};

struct Status
{
    State completion_state;
    boolean warning;
    string status_message;
};
```

```
struct EntityNode
{
    NodeID id;
    string entity_name;
};

struct EntityRelationship
{
    NodeID start_node;
    NodeID end_node;
    Cardinality start_to_end_card;
    Cardinality end_to_start_card;
};

typedef sequence < EntityNode > Entity NodeList;
typedef sequence < EntityRelationship > Entity RelationshipList;

struct EntityGraph
{
    Entity NodeList nodes;
    Entity RelationshipList relationship;
};

// *****
//   Exception Structure and Exceptions for the UCO (also
//   used for GIAS, GIXS and Profile Service)
// *****

struct exception_details {
    string exception_name;
    boolean standard_exception_name;
    string exception_desc;
};
```

UNCLASSIFIED

```
//InvalidInputParameter Exception

exception InvalidInputParameter {
    exception_details details;
    UCO::NameList badInputParameters;
};

//Processing Fault Exception

exception ProcessingFault {
    exception_details details;
};

//System Fault Exception

exception SystemFault {
    exception_details details;
};

};      // End of module UCO
```

Appendix B: PS IDL

```
*****  
// FILE:           profile.idl  
//  
//  
//  
// DESCRIPTION:     Profile Service  
//  
//      Defines the data types and interfaces needed to support  
// search,  
//      retrieval and access to user, node, and system profiles.  
//  
// LIMITATIONS:  
//  
//  
// SOFTWARE HISTORY:  
//  
//<  
*****  
  
#include "uco.idl"  
  
#include "gias.idl"  
  
*****  
// MODULE: PS  
//>   The main module for the Profile Service  
//<  
*****  
  
module PS  
{
```

```
/* **** */
/*
 * The Exception Identifiers for Profile Service (PS) Module
 * **** */

const string UnknownProfileElementTypeConst =
"UnknownProfileElementType";
const string BadAccessCriteriaConst = "BadAccessCriteria";
const string BadProfileElementConst = "BadProfileElement";
const string BadAccessValueConst = "BadAccessValue";
const string PasswordExpiredConst = "PasswordExpired";
const string InvalidOrderConst = "InvalidOrder";
const string InvalidQueryConst = "InvalidQuery";
const string UnknownOperationConst = "UnknownOperation";
const string VolumeReadAccessDeniedConst =
"VolumeReadAccessDenied";
const string VolumeWriteAccessDeniedConst =
"VolumeWriteAccessDenied";
const string FolderNotEmptyConst = "FolderNotEmpty";
const string FolderExistsConst = "FolderExists";
const string UnknownFolderItemTypeConst = "UnknownFolderItemType";
const string UnknownVolumeConst = "UnknownVolume";
const string UnknownPRIDConst = "UnknownPRID";
const string PermissionDeniedConst = "PermissionDenied";
const string BadFileLocationConst = "BadFileLocation";
const string SystemFaultConst = "SystemFault";

typedef string ProfileElementType;

typedef sequence <ProfileElementType> ProfileElementTypeList;

/* ***
```

```
// Forward References
// ***
interface ProfileMgr;
interface Profile;
interface ProfileElement;

// ***
// Forward References to the specific ProfileElements
// ***
interface BasicProfile;
interface GIASProfile;
interface SecureProfile;
interface SecureViewProfile;

// *****
*****  
// INTERFACE: ProfileMgr
//> The ProfileMgr provides access control to Profile objects
//<

// *****
*****  
interface ProfileMgr
{
    Profile get_profile(in UCO::NameValueList access_criteria)
        raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
    //> Returns the Profile for the user identified in
access_criteria.
//<

    string get_version()
        raises (UCO::ProcessingFault, UCO::SystemFault);
```

UNCLASSIFIED

```
//> Returns the version of this Profile manager.  
//<  
  
void set_user_password  
    (in UCO::NameValueList access_criteria,  
     in string new_password)  
    raises (UCO::InvalidInputParameter,  
           UCO::ProcessingFault, UCO::SystemFault);  
  
};  
  
//*****  
*****  
// INTERFACE: Profile  
//> The Profile object provides access to the contents of a  
Profile  
// by serving as a container for ProfileElement objects of  
different types.  
//<  
  
//*****  
*****  
interface Profile  
{  
    ProfileElementTypeList get_profile_element_types()  
        raises (UCO::ProcessingFault, UCO::SystemFault);  
  
    //> Returns a list of all ProfileElementTypes contained in  
this Profile.  
    //<  
  
    ProfileElement      get_profile_element(in ProfileElementType  
element_type)  
        raises (UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);
```

```
//> Returns the ProfileElement of the specified type.  
//<  
  
SecureProfile      get_secure_profile_element  
                   (in UCO::NameValueList  
trusted_access_criteria)  
                   raises (UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
//> This operation returns the secure profile element that  
contains  
  
// user security information that does not change across  
views.  
  
// The trusted access criteria limits the availability of  
this  
  
// information. If the access criteria does not contain  
expected names,  
  
// the operation will return a BadAccessCriteria exception  
identifier. If the access  
  
// criteria does not contain expected values, the operation  
will return a  
  
// BadAccessValue exception identifier.  
//<  
  
SecureViewProfile  get_secure_view_profile_element  
                   (in UCO::NameValueList  
trusted_access_criteria,  
                   in GIAS::ViewName      view)  
                   raises (UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
//> This method returns the secure profile element that  
contains  
  
// user security information that is view specific. The  
// trusted access criteria limits the availability of this  
  
// information. If the access criteria does not contain  
expected names,  
  
// the operation will return a BadAccessCriteria exception  
identifier. If the access
```

UNCLASSIFIED

```
// criteria does not contain expected values, the operation
will return a
// BadAccessValue exception identifier.
//<

UCO::AbsTime      get_last_update_time()
raises (UCO::ProcessingFault, UCO::SystemFault);
//> Returns the time the Profile was last changed.
//<

void            get_profiled_views(out UCO::NameList
view_list)
raises (UCO::ProcessingFault, UCO::SystemFault);
//> Returns a list of views that are valid for a particular
profile
//<

};

//*****
// INTERFACE ProfileElement
//> The ProfileElement object serves as the base abstract class
for all
// types of content objects in a Profile. It contains
operations common to
// all types of ProfileElement objects.
//<

//*****
interface ProfileElement
{
```

```
UCO::AbsTime get_last_update_time()
    raises (UCO::ProcessingFault, UCO::SystemFault);
    //> Returns the time this ProfileElement was last changed.
    //<
};

// *****
// INTERFACE: BasicProfile
//
//> The BasicProfile object is a specialization of the
ProfileElement
// object to support all types of users. It contains operations
and
// types common to all types of users.
//<

// *****
interface BasicProfile : ProfileElement
{

    struct TelephoneNumber
    {
        string name;
        string number;
    };

    typedef sequence<TelephoneNumber> TelephoneNumberList;

    struct UserInformation
{



UNCLASSIFIED
```

```
        string          name;
        string          organization;
        string          address;
        string          city;
        string          state;
        string          zip;
        string          country;
        string          email;
        TelephoneNumberList phone_numbers;
        UCO::FileLocation    ftp_location;
        UCO::AbsTime        password_expiration;

    };

struct UserPreference
{
    string  name;
    string  value;
    boolean editable;
    string  description;
};

typedef sequence<UserPreference> UserPreferenceList;

typedef sequence<string> PreferenceNameList;

struct UserPreferenceDomain
{
    GIAS::Domain adomain;
    boolean      multi_select;
};


```

```
struct SecurityInformation
{
    string classification;
    boolean security_administrator_flag;
};

void get_user_information      (out UserInformation      info)
raises (UCO::ProcessingFault, UCO::SystemFault);

void set_user_information      (in  UserInformation      info)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void get_security_information  (out SecurityInformation info)
raises (UCO::ProcessingFault, UCO::SystemFault);

void get_available_preferences (out PreferenceNameList names)
raises (UCO::ProcessingFault, UCO::SystemFault);

void get_preference_domain     (in  string
preference_name,
                                out GIAS::Domain
domain)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void get_user_preference       (in  string
preference_name,
                                out UserPreference
preference)
raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

void get_user_preferences      (out UserPreferenceList  list)
```

UNCLASSIFIED

```

        raises (UCO::ProcessingFault, UCO::SystemFault);

        void set_user_preference      (in UserPreference
preference)
            raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

        void set_user_preferences    (in UserPreferenceList
preferences)
            raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

    } ;

// ****
*****  

// INTERFACE: SecureProfile
//
//> The SecureProfile object is a specialization of the
ProfileElement
// object to support basic security information access.
//<

*****  

interface SecureProfile : ProfileElement
{
    // ***
    // Authorization Information
    // ***

    struct Authorization
    {
        string      authorization_classification;

```

UNCLASSIFIED

```
UCO::NameList exception_country_list;
UCO::NameList releasable_country_list;
UCO::NameList releasable_org_list;
UCO::NameList distribution_limit_code_list;
UCO::NameList access_agreement_list;
UCO::NameList compartment_list;
};

//> This data structure holds the elements that constitute
// a discretionary access authorization for the user.
//<

void get_authorization
    (out Authorization aauthorization)
raises (UCO::ProcessingFault, UCO::SystemFault);
//> This operation returns the authorization information.
//<

void get_authorization_classification
    (out string classification)
raises (UCO::ProcessingFault, UCO::SystemFault);
//> This operation returns a user's authorization
classification level.
//<

void get_exception_by_countries
    (out UCO::NameList country_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

//> This operation returns the exception by country list.
//<

void get_releasable_countries
    (out UCO::NameList country_list)
```

UNCLASSIFIED

```
    raises (UCO::ProcessingFault, UCO::SystemFault);

    //> This operation returns the releasable country list.
    //<

    void      get_releasable_organizations
              (out UCO::NameList      organization_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);

    //> This operation returns the releasable organization list.
    //<

    void      get_distribution_limitation_codes
              (out UCO::NameList      code_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);

    //> This operation returns the distribution limitation code
list.
    //<

    void      get_access_agreements
              (out UCO::NameList      agreement_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);

    //> This operation returns the access agreement list.
    //<

    void      get_compartments
              (out UCO::NameList      compartment_list)
    raises (UCO::ProcessingFault, UCO::SystemFault);

    //> This operation returns the compartment list for the user.
    //<
```

UNCLASSIFIED

```
 //***  
 // Attribute/entity Restriction Information  
 /***  
  
 enum AccessType {READ_DENIED, WRITE_DENIED};  
  
 void get_restricted_attributes  
     (in AccessType access_type,  
      out UCO::NameList attribute_list)  
 raises (UCO::InvalidInputParameter, UCO::ProcessingFault,  
 UCO::SystemFault);  
 //> This operation returns the restricted attribute  
 identifiers  
 // for the user relative to a specific view.  
 //<  
  
 void get_restricted_entities  
     (in AccessType access_type,  
      out UCO::NameList entity_list)  
 raises (UCO::InvalidInputParameter, UCO::ProcessingFault,  
 UCO::SystemFault);  
 //> This operation returns the restricted entity identifiers  
 // for the user relative to a specific view.  
 //<  
  
 } ;  
  
 //*****  
 *****  
 // INTERFACE: SecureViewProfile  
 //
```

```
//> The SecureViewProfile object is a specialization of the
SecureProfile
//   object to support view-oriented security information access.
//<

//*****
*****  
interface SecureViewProfile : SecureProfile
{
    //***  
    // Authorization Information  
    //***  
  
    boolean use_authorization()  
    raises (UCO::ProcessingFault, UCO::SystemFault);  
  
    //> This operation returns a true indication if the
authorizaton
    //   is to be used for discretionary access control for
    //   the specified view.
    //<  
  
    //***  
    // Attribute Value Restriction Information  
    //***  
  
    void get_restricted_attribute_values
        (out string      restriction)
    raises (UCO::ProcessingFault, UCO::SystemFault);
    //> This operation returns the restricted attribute value
    //   string for the user relative to a specific view.
    //<
};
```

UNCLASSIFIED

```
//*****
*****  
// INTERFACE: GIASProfile  
//  
//> The GIASProfile object is a specialization of the  
ProfileElement  
// object to support users of GIAS Libraries. It contains  
methods and  
// types specific to GIAS libraries operations.  
//<  
  
//*****  
*****  
interface GIASProfile : ProfileElement  
{  
  
    //***  
    // Defines a profile id used to uniquely identify objects  
within the  
    // user profile  
    //***  
    typedef string          PRID;  
    typedef sequence <PRID> PRIDList;  
  
    void get_allowable_operations(out UCO::NameList  
operation_list)  
        raises (UCO::ProcessingFault, UCO::SystemFault);  
        //> This operation returns a list of the allowable operations  
for  
        // a particular user.  
    //<
```

UNCLASSIFIED

```
boolean operation_is_allowed (in string operation)
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
    //> This operation returns TRUE if the specified manager is
    // accessible by the user, otherwise FALSE is returned.
    //<

typedef string                               FolderTree;
// string defined as: \folder1\folder2\folder3\etc...
// An empty string or '\' denotes the root folder.

struct FolderLocation
{
    string      volume;
    FolderTree   folder;
};

struct VolumeInfo
{
    boolean      default_volume;
    string       volume;
    boolean      read_permission;
    boolean      write_permission;
    boolean      create_delete_permission;
};

typedef sequence < VolumeInfo > VolumeInfoList;
typedef sequence < FolderLocation> FolderLocationList;

enum FolderItemType
{
    SAVED_QUERY,
```

UNCLASSIFIED

```
SUBMITTED_QUERY,  
SUBMITTED_QUERY_AND_HIT_COUNT,  
  
STANDING_QUERY,  
SAVED_ORDER,  
SUBMITTED_ORDER,  
STANDING_ORDER,  
SUBMITTED_CREATE,  
SUBMITTED_HIT_COUNT,  
RESULTS_DIGEST,  
SESSION,  
ALL  
};  
  
struct FolderItem  
{  
    PRID           item_id;  
    FolderLocation location;  
    string          name;  
    string          description;  
    FolderItemType type;  
    UCO::AbsTime   creation_time;  
    UCO::AbsTime   last_accessed_time;  
    UCO::AbsTime   last_modified_time;  
    string          owner_name;  
    string          user_created_name;  
    string          user_last_accessed_name;  
    string          user_last_modified_name;  
    UCO::Rectangle area_of_interest_mbr;  
};  
  
typedef sequence < FolderItem >    FolderItemList;  
UNCLASSIFIED
```

```
// ***
// FolderItem holds one of the following:
//   SubmittedQuery
//   SubmittedQueryAndHitCount
//   SubmittedOrder
//   SubmittedCreate
//   StandingQuery
//   StandingOrder
//   SavedQuery
//   SavedOrder
//   ResultsDigest
//   SavedSession
// ***
typedef any FolderContent;
typedef sequence <FolderContent> FolderContentList;

enum SearchDepth
{
    SINGLE_FOLDER,
    FOLDER_TREE
};

// ***
// Return a list of available volumes
// ***
void list_volumes(
    out VolumeInfoList volume_list)
raises (UCO::ProcessingFault, UCO::SystemFault);

// ***
// Creates a new folder in the location specified
// ***
```

UNCLASSIFIED

```
void new_folder (
    in FolderLocation folder)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

// ***
// Moves a folder's location
// ***
void update_folder (
    in FolderLocation folder,
    in FolderLocation new_location)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

// ***
// Removes an empty folder
// ***
void remove_folder (
    in FolderLocation folder)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

// ***
// Return a list of folders in a specific folder
// ***
void list_folders (
    in FolderLocation      folder,
    out FolderLocationList query_items)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);
```

UNCLASSIFIED

```
 // ***
// Update a specific folder item in the profile
// ***
void update_folder_item (
    in PRID           item_id,
    in FolderLocation location,
    in string          name,
    in string          description,
    in UCO::Rectangle area_of_interest_mbr)
    raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

 // ***
// Returns the contents of a folder item
// N.B. Some folder item types may override this operation.
// ***
void get_entry (
    in PRID           item_id,
    out FolderContent entry)
    raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

 // ***
// Removes a folder item
// ***
void remove_entry (
    in PRID           item_id)
    raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

UNCLASSIFIED

```
/***
// Returns a list of folder contents of a given type.
// Can search just in a given folder or recursively search
the folder
// tree starting at a given folder location.
// N.B. This call may not be useful for some folder item
types.

/***
void list_entries (
    in FolderItemType      type,
    in FolderLocation      starting_point,
    in SearchDepth         depth,
    out FolderContentList  entries)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);

/***
// Returns a list of folder information for a given folder
item type.
// Can search just in a given folder or recursively search
the folder
// tree starting at a given folder location.
/***
void list_entry_items (
    in FolderItemType      type,
    in FolderLocation      starting_point,
    in SearchDepth         depth,
    out FolderItemList     entry_items)
raises (
    UCO::InvalidInputParameter, UCO::ProcessingFault,
    UCO::SystemFault);
```

```
/* ****
****

    // Submitted Queries
    //
    //> These are queries that have been submitted to a Library,
but the
    // client has not yet completed the retrieval of the query
results.
    // This is useful when queries are time-consuming.
//<

/* ****
****

    struct SubmittedQuery
    {
        FolderItem           item_info;
        GIAS::SubmitQueryRequest request;
    };
    typedef sequence<SubmittedQuery> SubmittedQueryList;

    struct SubmittedQueryAndHitCount
    {
        FolderItem           item_info;
        GIAS::SubmitQueryRequest request;
        GIAS::RequestList    hit_count_requests;
    };
    typedef sequence<SubmittedQueryAndHitCount>
SubmittedQueryAndHitCountList;

PRID new_submitted_query (
    UNCLASSIFIED
```

```

        in FolderLocation           folder,
        in string                  name,
        in string                  description,
        in UCO:::Rectangle          area_of_interest_mbr,
        in GIAS:::SubmitQueryRequest request)

    raises (
        UCO:::InvalidInputParameter, UCO:::ProcessingFault,
        UCO:::SystemFault);

PRID new_submitted_query_and_hitcount (
        in FolderLocation           folder,
        in string                  name,
        in string                  description,
        in UCO:::Rectangle          area_of_interest_mbr,
        in GIAS:::SubmitQueryRequest request,
        in GIAS:::RequestList       hit_count_requests)

    raises ( UCO:::InvalidInputParameter, UCO:::ProcessingFault,
        UCO:::SystemFault);

struct SubmittedHitCount
{
    FolderItem                  item_info;
    GIAS:::HitCountRequest      request;
};

typedef sequence<SubmittedHitCount> SubmittedHitCountList;

PRID new_submitted_hitcount (
        in FolderLocation           folder,
        in string                  name,
        in string                  description,
        in UCO:::Rectangle          area_of_interest_mbr,

```

```
        in GIAS::HitCountRequest      request)
           raises (UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

//*****  
*****
// Submitted Orders
//
//> These are orders that have been submitted to a Library,
but the
// user has not yet deleted. A user may need to keep orders
around
// if they take a long time to complete.
//<

//*****  
*****
struct SubmittedOrder
{
    FolderItem                  item_info;
    GIAS::OrderRequest          request;
};

typedef sequence<SubmittedOrder> SubmittedOrderList;

PRID new_submitted_order(
    in FolderLocation       folder,
    in string                 name,
    in string                 description,
    in UCO::Rectangle         area_of_interest_mbr,
    in GIAS::OrderRequest    order)
    raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);
```

```
//*****
*****  
  
    // Submitted Creates  
    //  
    //> These are creates that have been submitted to a Library,  
but the  
    // user has not yet deleted. A user may need to keep these  
around  
    // if they take a long time to complete.  
    //<  
  
//*****  
*****  
  
struct SubmittedCreate  
{  
    FolderItem           item_info;  
    GIAS::CreateRequest request;  
};  
typedef sequence<SubmittedCreate> SubmittedCreateList;  
  
PRID new_submitted_create(  
    in FolderLocation      folder,  
    in string               name,  
    in string               description,  
    in UCO::Rectangle       area_of_interest_mbr,  
    in GIAS::CreateRequest create_request)  
    raises (  
        UCO::InvalidInputParameter, UCO::ProcessingFault,  
        UCO::SystemFault);  
  
*****
```

```
// Standing Queries
//
//> These are queries that get executed on a scheduled basis.
The client
// needs to be able to access the query request at any time.
//<

//*****
*****  
  
struct StandingQuery
{
    FolderItem           item_info;
    GIAS::SubmitStandingQueryRequest request;
};  
  
typedef sequence <StandingQuery> StandingQueryList;  
  
PRID new_standing_query (
    in FolderLocation           folder,
    in string                   name,
    in string                   description,
    in UCO::Rectangle
area_of_interest_mbr,
    in GIAS::SubmitStandingQueryRequest request)
    raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);  
  
//*****
*****  
  
// Standing Orders
//
```

```
//> These are orders that get executed on a scheduled basis.  
The client  
// needs to be able to access the order request at any time.  
//<  
  
//*****  
*****  
struct StandingOrder  
{  
    FolderItem           item_info;  
    GIAS::QueryOrderContents   order;  
    GIAS::SubmitStandingQueryRequest query;  
};  
  
typedef sequence <StandingOrder> StandingOrderList;  
  
PRID new_standing_order(  
    in FolderLocation          folder,  
    in string                  name,  
    in string                  description,  
    in GIAS::QueryOrderContents  
order,  
    in UCO::Rectangle          area_of_interest_mbr,  
    in GIAS::SubmitStandingQueryRequest query)  
    raises (UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
//*****  
*****  
// Saved Queries  
//  
//>
```

```
//<

// *****
struct SavedQuery
{
    FolderItem           item_info;

    GIAS::Query          bqs;
    UCO::FileLocation    thumbnail_location;
    boolean               browse_image_returned_flag;
    UCO::NameList         result_attributes;

    GIAS::SortAttributeList sort_attributes;
    string                geographic_datum;
    UCO::AbsTime          last_submitted_date;
};

typedef sequence < SavedQuery >   SavedQueryList;

// ***
// Save the query information to the specified folder.
// ***
PRID new_saved_query (
    in FolderLocation           folder,
    in string                     name,
    in string                     description,
    in GIAS::Query                bqs,

    in boolean                    browse_image_returned_flag,
```

UNCLASSIFIED

```
    in UCO::NameList          result_attributes,
    in GIAS::SortAttributeList sort_attributes,
    in string                  geographic_datum,
    in UCO::AbsTime            last_submitted_date,
    in UCO::Rectangle          area_of_interest_mbr)

raises ( UCO::InvalidInputParameter, UCO::ProcessingFault,
SystemFault);

// ***
// Update a specified query from the profile
// ***
void update_saved_query(
    in PRID                 saved_query_id,
    in string                name,
    in string                description,
    in GIAS::Query           bqs,

    in boolean               browse_image_returned_flag,
    in UCO::NameList          result_attributes,
    in GIAS::SortAttributeList sort_attributes,
    in string                  geographic_datum,
    in UCO::AbsTime            last_submitted_date,
    in UCO::Rectangle          area_of_interest_mbr)

raises (
UCO::InvalidInputParameter, UCO::ProcessingFault,
UCO::SystemFault);

***** ****
// Saved Orders
//
//>
//<
```

UNCLASSIFIED

```
//*****
*****  
  
struct SavedOrder  
{  
    FolderItem      item_info;  
    GIAS::OrderContents      order;  
};  
  
typedef sequence < SavedOrder > SavedOrderList;  
  
PRID new_saved_order (  
    in FolderLocation      folder,  
    in string              name,  
    in string              description,  
    in UCO::Rectangle      area_of_interest_mbr,  
    in GIAS::OrderContents      order)  
    raises (  
UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
void update_saved_order(  
    in PRID                saved_order_id,  
    in string              name,  
    in string              description,  
    in UCO::Rectangle      area_of_interest_mbr,  
    in GIAS::OrderContents      order)  
    raises (  
UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);
```

```
//*****
*****  
  
    // Results Digests  
    //  
    //>  
    //<  
  
//*****  
*****  
  
    struct ResultsDigest  
{  
        FolderItem      item_info;  
    };  
  
    typedef sequence < ResultsDigest >    ResultsDigestList;  
  
//***  
// Save the results digest to the specified folder... the  
// server will ftp get the digest from the specified file  
// location.  
//***  
PRID new_results_digest (   
    in FolderLocation      folder,  
    in string              name,  
    in string              description,  
    in UCO::Rectangle      area_of_interest_mbr,  
    in UCO::FileLocation   digest_location)  
    raises (   
        UCO::InvalidInputParameter, UCO::ProcessingFault,  
        UCO::SystemFault);  
  
//***  
// Update the specified results digest in the profile
```

UNCLASSIFIED

```
/***
void update_results_digest(
    in PRID                  results_digest_id,
    in string                 name,
    in string                 description,
    in UCO:::Rectangle         area_of_interest_mbr,
    in UCO:::FileLocation     digest_location)
raises (
    UCO:::InvalidInputParameter, UCO:::ProcessingFault,
    UCO:::SystemFault);

/***
// Get the results digest for the specified ID... the server
// will ftp put the digest to the specified file location.
/***
void get_results_digest (
    in PRID                  query_id,
    in UCO:::FileLocation     destination_file)
raises (
    UCO:::InvalidInputParameter, UCO:::ProcessingFault,
    UCO:::SystemFault);

***** ****
***** // Saved Sessions
//>
//<
***** ****

struct SavedSession
```

```
{  
    FolderItem      item_info;  
    string          session_text;  
};  
  
typedef sequence < SavedSession >  SavedSessionList;  
  
//***  
// Save the session to the specified folder... the  
// server will ftp get the session from the specified file  
// location.  
// ***  
PRID new_saved_session (  
    in FolderLocation      folder,  
    in string              name,  
    in string              description,  
    in UCO::Rectangle      area_of_interest_mbr,  
    in UCO::FileLocation   session_location)  
raises (  
    UCO::InvalidInputParameter, UCO::ProcessingFault,  
    UCO::SystemFault);  
  
//***  
// Update the specified saved session in the profile  
//***  
void update_saved_session(  
    in PRID                saved_session_id,  
    in string              name,  
    in string              description,  
    in UCO::Rectangle      area_of_interest_mbr,  
    in UCO::FileLocation   session_location)  
raises (
```

```
UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
//***  
// Get the saved session for the specified ID... the server  
// will ftp put the session to the specified file location.  
//***  
void get_saved_session (  
    in PRID                 query_id,  
    in UCO::FileLocation destination_file)  
raises (  
UCO::InvalidInputParameter, UCO::ProcessingFault,  
UCO::SystemFault);  
  
}; // end interface GIASProfile  
  
}; // end module PS
```

Appendix D: Universal Product Identification

```
// ****
// *
// *      The USIGS Universal Product Identifier
// *      Specification
// *
// *
// *      Description: Defines a universal USIGS product
// *      identification.
// *
// *
// *      History:
// *      Date          Author    Comment
// *      ----          -----    -----
// *      29 Sept 98    J. Baldo and D. Lutz  Initial release
for
// *                                review.
// *
// *      Notes
// *      -----
// *      NONE
// *
// *
// */
module UID
{
interface Product
{
};

typedef sequence < Product > ProductList;

};


```

UNCLASSIFIED

Appendix E: Encoding Rules for StringDAG

This Appendix defines the encoding rules to allow arbitrary IDL defined data types to be expressed as strings for use in the UCO::StringDAG datatype. This Appendix also defines a set of optimized encoding rules for specific commonly used UCOS datatypes. These optimized definitions take precedence of the more general used and must be used whenever that UCOS datatype is to be encoded.

All encodings take the general form of *ID < value >*, where ID identifies the type (or “OPT” for the optimized types, see below) which has been encoded and the value is the actual encoding of that type. The value is completely enclosed within the “< ... >” pair i.e the first non-whitespace character after the ID is always “<” and the last character in the whole string is always “>”.

CORBA Datatype Encodings

CORBA Type	Lexical Representation	Example	Null Value
Short	short <+ - n>	“short <-123>”	“short <>”
unsigned short	ushort <n>	“ushort<123>”	“ushort<>”
Long	long <+ - n>	“long <-123>”	“long <>”
unsigned long	ulong <n>	“ulong <123>”	“ulong <>”
long long	longlong <+ -n>	“longlong <-123>”	“longlong <>”
unsigned long long	ulonglong <n>	“ulonglong <123>”	“ulonglong <>”
Float	float < + - mantissa E + - exponent >	“float <-345.567>” “float <-3.4567 E 2>”	“float <>”
Double	double < + - mantissa E + - exponent >	“double <-345.567>” “double <-3.4567 E 2>”	“double <>”
long double	longdouble < + - mantissa E + - exponent >	“longdouble <-345.567>” “longdouble <-3.4567 E 2>”	“longdouble <>”
Char	char <c>	“char<a>”	“char<>”
Boolean	boolean < TRUE FALSE>	“boolean <TRUE>”	“boolean <>”
Octet	octet 	“octet <7>”	“octet <>”
Struct	struct <structName < fieldname <fieldvalue> fieldname <fieldvalue> > >	“struct < UCO::Date < year < ushort <1999 >> month < ushort <3>> day < ushort <25>> > >”	“struct < structName <>>”
Enum	enum< enumName <s> >	“enum<UCO::State <COMPLETED>>”	“enum<UCO::State <>>”
Sequence	sequence < sequenceName < sequenceType <elementvalue> <elementValue> > >	“sequence < GIAS::RsetList < short <1> <3> <7> > >”	“sequence < GIAS::RsetList <>>”
String	string <ssss >	“string <This is a note>”	“string <>”

UNCLASSIFIED

		“string < 3 < 4 >”	
IOR	ior<s>	ior<XXX:djfdkjif34553>	ior<>

- 1) The CORBA any, union, wchar, wstring, fixed, array and native data types do not currently have a encoding rule.

UCOS Specific Optimized Encodings

CORBA Type	Lexical Representation	Example	Null Value
UCO:Date	OPT< UCO:Date <YYYY/MM/DD>>	OPT < UCO:Date <2000/08/22> >	OPT < UCO:Date <>>
UCO:AbsTime	OPT< UCO:AbsTime <YYYY/MM/DD HH:MM:SS.S >>	OPT < UCO:AbsTime <2000/08/22 14:48:23.4> >	OPT < UCO:AbsTime <>>
UCO:Coordinate2D	OPT < UCO:Coordinate2D <lat coord,lon coord> >	OPT < UCO:Coordinate2D <37.43 , 32.32> >	OPT < UCO:Coordinate2D <>>
UCO:Coordinate3D	OPT < UCO:Coordinate3D <lat coord,lon coord, height > >	OPT < UCO:Coordinate3D <37.43 , 32.32, 345.6> >	OPT < UCO:Coordinate3D <>>
UCO:Polygon	OPT < UCO:Polygon <lat coord,lon coord,lat coord,lon coord, lat coord,> >	OPT < UCO:Polygon < 35.25 ,16.10 35.47,16.10,35.47,16.08, 35.25,16.08> >	OPT < UCO:Polygon <>>
UCO:Rectangle	OPT < UCO:Rectangle< UL_lat coord, UL_lon coord, LR_lat coord,LR_lon coord > >	OPT < UCO:Rectangle <35.25,16.10,35.47,16.08 > >	OPT < UCO:Rectangle <>>

UNCLASSIFIED

UNCLASSIFIED

Appendix F: Reference OMG Standard IDL

CORBA Standard Exceptions

```
#define ex_body {unsigned long minor; completion_status completed; }

enum completion_status {COMPLETED_YES, COMPLETED_NO,
COMPLETED_MAYBE};

enum exception_type {NO_EXCEPTION, USER_EXCEPTION,
SYSTEM_EXCEPTION};

exception UNKNOWN ex_body;
exception BAD_PARAM ex_body;
exception NO_MEMORY ex_body;
exception IMP_LIMIT ex_body;
exception COMM_FAILURE ex_body;
exception INV_OBJREF ex_body;
exception NO_PERMISSION ex_body;
exception INTERNAL ex_body;
exception MARSHAL ex_body;
exception INITIALIZE ex_body;
exception NO_IMPLEMENT ex_body;
exception BAD_TYPECODE ex_body;
exception BAD_OPERATION ex_body;
exception NO_RESOURCES ex_body;
exception NO_RESPONSE ex_body;
exception PERSIST_STORE ex_body;
exception BAD_INV_ORDER ex_body;
exception TRANSIENT ex_body;
exception FREE_MEM ex_body;
exception INV_IDENT ex_body;
exception INV_FLAG ex_body;
exception INTF_REPO ex_body;
```

UNCLASSIFIED

```
exception BAD_CONTEXT ex_body;
exception OBJ_ADAPTER ex_body;
exception DATA_CONVERSION ex_body;
exception OBJECT_NOT_EXIST ex_body;
```

UNCLASSIFIED

Appendix G: Acronyms

API	Application Program Interface
CIIF	Common Imagery Interoperability Facilities
CIIP	Common Imagery Interoperability Profile
CIIWG	Common Imagery Interoperability Working Group
CORBA	Common Object Request Broker Architecture
GIAS	Geospatial & Imagery Access Services
IASS	Image Access Services Specification
IDL	Interface Definition Language
ISO	International Standard Organization
NIMA	National Imagery and Mapping Agency
OGC	Open GIS Consortium
OMG	Object Management Group
TBD	To Be Determined
TBR	To Be Resolved
UCOS	USIGS Common Object Specification
UIP	USIGS Interoperability Profile
USIGS	United States Imagery and Geospatial Service

Appendix H: Points of Contact

NIMA/ATSR

Ron Burns, National Imagery and Mapping Agency

Phone: 703.755.5630

Email: BurnsR@nima.mil

NIMA/ATSRI

Bill Young, National Imagery and Mapping Agency

Phone: 703.755.5644

Email: YoungW@nima.mil

USIGS Interface Definition & Implementation

Charlie Green, SI, Engineering Edge Alliance (Sierra Concepts, Inc.)

Phone: 610.347.0602

Email: cpg.sci@mindspring.com

UCOS & GIAS Specifications, RFCs & Support

Dave Lutz, The MITRE Corporation

Phone: 703.883.7848

Email: dlutz@mitre.org

USIGS Interoperability Profile (UIP)

Bradley Bretzin, SI, Engineering Edge Alliance (Booz•Allen & Hamilton)

Phone: 703.375.2034

Email: bretzinb@bah.com