IMS Enterprise Services Best Practice and Implementation Guide

Version 1.0 Final Specification

Document Name: IMS Enterprise Services Best Practice and Implementation Guide

Date: 11 June 2004

Table of Contents

1. Introduction

1.1 Enterprise Services Overview

The Enterprise Services specification [EntService, 04b] is the definition of how systems manage the exchange of information that describes people, group and memberships within the context of learning. The Enterprise Services specification is constructed following the recommendations documented in the IMS Abstract Framework (IAF) [AbsGloss, 03], [AbsASC, 03], [AbsWhite, 03]. This means that this specification is based upon:

- Interoperability – Enterprise Services focuses on the exchange of information between Enterprise systems. The specification makes no assumptions on how the data is managed within the Enterprise systems;
- Service-oriented – Enterprise Services defines the exchange of information in terms of the services being supplied by the collaboration of the systems. This takes the form of Person Management Services, Group Management Services and Membership Management Services;
- Component-based – the set of services will be supplied such that they can be combined to form a range of services. The Person Management Services, Group Management Services and Membership Management Services can be combined to provide other services and the Enterprise Service will have other services added to it in later releases;
- Layering – the Enterprise Service and its constituent services (Person, Group and Membership) are part of the Application Services layer;
- Behaviors and Data Models – the Enterprise Services are defined in terms of their behaviors and data models. The behaviors cause changes in the state of the data model and the state of the data model will only be altered as a result of a clearly defined behavior;
- Multiple Bindings – the Enterprise Services information model is to be defined using the Unified Modelling Language (UML). This enables reliable mapping of the information model into a range of different bindings. The bindings of immediate importance are to the Web Services Description Language (WSDL);
- Adoption – the Enterprise Services are based upon the original Enterprise specification data model. While there are significant changes the underlying data model has been maintained and the core Person, Group and Membership structures remain.

1.2 Scope and Context

This document is the IMS Enterprise Services Best Practice and Implementation Guide v1.0 and as such it is used to support the following documents:

a) IMS Enterprise Services Core Use Cases v1.0 [EntService, 04a] – the set of use cases that are the basis for the definition of the information model;

b) IMS Enterprise Services Specification v1.0 [EntServices, 04b] – this presents the overall structure and capabilities of the Enterprise Services specification;

c) IMS Enterprise Services Conformance Specification v1.0 [EntServices, 04c] – the definition of the conformance criteria that must be followed by systems that wish to claim compliance to the Enterprise Services Information model;

d) IMS Person Management Services Information Model v1.0 [PersonServices, 04a] – the information model of the Person Management Services;

e) IMS Person Management Services WSDL Binding v1.0 [PersonServices, 04b] – the description of the WSDL binding of the Person Management Services information model;

f) IMS Group Management Services Information Model v1.0 [GroupServices, 04a] – the information model of the Group Management Services;

g) IMS Group Management Services WSDL Binding v1.0 [GroupServices, 04b] – the description of the WSDL binding of the Group Management Services information model;

h) IMS Membership Management Services Information Model v1.0 [MemberServices, 04a] – the information model of the Membership Management Services;

i) IMS Membership Management Services WSDL Binding v1.0 [MemberServices, 04b] – the description of the WSDL binding of the Membership Management Services information model.

As such the Enterprise Services specification supersedes the original Enterprise specifications:

a) IMS Enterprise Information Model Final Specification v1.1 [Enterprise, 02a].

b) IMS Enterprise XML Binding Final Specification v1.1 [Enterprise, 02b];

c) IMS Enterprise Services Best Practice and Implementation Guide Final Specification v1.0 [Enterprise, 02c].

This best practice and implementation guide describes some of the issues that need to be addressed during various stages of adoption. This is the first version of these services and so most of the best practice is derived from experience of proprietary implementations of similar functionality.

1.3 Structure of this Document

The structure of this document is:

2. Related Specifications and Activities	The relationship of this specification activity to other IMS and external specification activities;
3. Overall Services Model	A brief summary of the IMS Enterprise Services information model and WSDL binding;
4. Example SOAP/HTTP Messages for Synchronous Services	Examples of the basic Synchronous Enterprise Services messages that are exchanged assuming SOAP/HTTP as the transport;
5. Example SOAP/HTTP Messages for Asynchronous Services	Examples of the basic Asynchronous Enterprise Services messages that are exchanged assuming SOAP/HTTP as the transport;
6. Mapping for Other Specifications	The ways in which the IMS Enterprise Services functionality can be used to support other similar specifications;
7. Best Practice	Implementation guidance on the best practices to be adopted when using the Enterprise Service specification;
8. Supporting the Use Cases	A brief explanation of how the use cases defined in the Enterprise Services Core Uses-cases can be supported using the Enterprise Services specification;
9. Extending the Services	A brief explanation of the ways in which the Enterprise Services can be extended without causing problems with backwards compatibility. This material will also give an indication of the ways in which the Enterprise Services are to be extended in later versions;
Appendix A – Glossary of Terms	A glossary of the key terms and elements used within the specification.
Appendix B – OKI Enterprise Open Service Interface Definitions	A brief description of the OKI Enterprise Services OSIDs that are used to support the explanation of the usage of the Enterprise Services as an implementation binding based upon the OKI.

1.4 Nomenclature

ADL	Advanced Distributed Learning
AICC	Aviation Industry CBT Committee
ANSI	American National Standards Institute
API	Application Programming Interface

CBT	Computer Based Training
CEN	Central European Normalization
DTD	Document Type Definition
EDI	Electronic Data Interchange
GWS	General Web Services
HR	Human Resources
HTTP	HyperText Transfer Language
IAF	IMS Abstract Framework
IETF	Internet Engineering Task Force
I-GMS	IMS Group Management Service
I-MMS	IMS Membership Management Service
I-PMS	IMS Person Management Service
ISO	International Standards Organization
LDAP	Lightweight Directory Access Protocol
LIP	Learner Information Package
OSID	Open Service Interface Definition
PDI	Personal Data Interchange
SCORM	Sharable Content Object Reference Model
SIF	Schools Interoperability Framework
SOAP	Simple Object Access Protocol
SOAPwA	Simple Object Access Protocol with Attachments
TEISS	Telematics European Industry Standardisation Support
UML	Unified Modelling Language
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
XML	Extensible Mark-up Language

1.5 References

[AbsASCs, 03]	*IMS Abstract Framework: Applications, Services & Components v1.0*, Ed. C.Smythe, IMS Global Learning Consortium, Inc., July 2003.
[AbsGloss, 03]	*IMS Abstract Framework: Glossary v1.0*, Ed. C.Smythe, IMS Global Learning Consortium, Inc., July 2003.
[AbsWhite, 03]	*IMS Abstract Framework: White Paper v1.0*, Ed. C.Smythe, IMS Global Learning Consortium, Inc., July 2003.
[Enterprise, 02a]	*IMS Enterprise Information Model Final Specification v1.1*, G.Collier, C.Etesse, W.Veres and C.Smythe, IMS Global Learning Consortium, Inc., July 2002.
[Enterprise, 02b]	*IMS Enterprise XML Binding Final Specification v1.1*, G.Collier, C.Etesse, W.Veres and C.Smythe, IMS Global Learning Consortium, Inc., July 2002.
[Enterprise, 02c]	*IMS Enterprise Best Practice & Implementation Guide Final Specification v1.1*, G.Collier, C.Etesse, W.Veres and C.Smythe, IMS Global Learning Consortium, Inc., July 2002.
[EntServices, 04a]	*IMS Enterprise Services Core Use Cases v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.
[EntServices, 04b]	*IMS Enterprise Services Specification v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.

[EntServices, 04c]	*IMS Enterprise Services Conformance Specification v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., Version 1.0, June 2004.
ESCommon, 04	*IMS Enterprise Services Common Data Specification v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., Version 1.0, June 2004.
[GroupServices, 04a]	*IMS Group Management Services Information Model v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.
[GroupServices, 04b]	*IMS Group Management Services WSDL Binding v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.
[GWS, 04a]	*IMS General Web Services Base Profiles Public Draft v1.0*, C.Schroeder, S.Raju, and C.Smythe, IMS Global Learning Consortium, Inc., Version 1.0, May 2004.
[GWS, 04b]	*IMS General Web Services Binding Methodology & Recipes Public Draft v1.0*, C.Schroeder, S.Raju, and C.Smythe, IMS Global Learning Consortium, Inc., Version 1.0, May 2004.
[LIP, 01a]	*IMS Learner Information Package Information Model v1.0*, R.Robson, C.Smythe, and F.Tansey, Version 1.0, IMS Global Learning Consortium Inc., March 2001.
[LIP, 01b]	*IMS Learner Information Package XML Binding v1.0*, R.Robson, C.Smythe, and F.Tansey, Version 1.0, IMS Global Learning Consortium Inc., March 2001.
[LIP, 01c]	*IMS Learner Information Packaging Best Practices & Implementation Guide v1.0*, R.Robson, C.Smythe, and F.Tansey, IMS Global Learning Consortium, Inc., March 2001.
[MemberServices, 04a]	*IMS Membership Management Services Information Model v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.
[MemberServices, 04b]	*IMS Membership Management Services WSDL Binding v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.
[OKI, 03]	*OKI Course management Open Service Interface Definition*, OKI, Doc Release v0.2, June 2003.
[PersonServices, 04a]	*IMS Person Management Services Information Model v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, Inc., June 2004.
[PersonServices, 04b]	*IMS Person Management Services WSDL Binding v1.0*, C.Smythe and C.Vento, IMS Global Learning Consortium, June 2004.
[SpecDev, 03]	*IMS Specification Development Methods & Best Practices Draft 5.0*, C.Smythe, IMS Global Learning Consortium, Inc., August 2003.
[VDEX, 04]	*IMS Vocabulary Definition Exchange Information Model v1.0*, A.Cooper and M.Mckell, IMS Global Learning Consortium, Inc., April 2004.

2. Related Specifications and Activities

2.1 IMS Enterprise v1.1 Specification

The IMS Enterprise Services specification supersedes the IMS Enterprise v1.1 specification. The services specification adds behaviors to the data model defined in the Enterprise v1.1 specification. The Enterprise Service uses a similar data model to that of the Enterprise v1.1 specification. All of the data-oriented information model features of the Enterprise v1.1 Specification are supported by the new Enterprise Services. The only core feature of the Enterprise v1.1 specification that is not supported is the <property> element. This element was used to provide some behavior capabilities and as such it is not required and is replaced by the usage of the service definitions.

2.2 Other IMS Specifications

2.2.1 IMS Learner Information Package v1.0 Specification

There is some overlap of functionality between the IMS Enterprise Services and the IMS Learner Information Package (LIP) specification [LIP, 01a], [LIP, 01b], [LIP, 01c]. The overlap is between the IMS Person Management Service (I-PMS) and the IMS LIP <identification> element. The I-PMS should be used for interactions that involve the IMS Group Management and IMS Membership Management services. The IMS LIP functionality should be used when working with individual profiles e.g., ePortfolios, life-long learning logs, etc.

2.2.2 IMS General Web Services Recommendations

The web service binding of the Enterprise Services specification is based upon the IMS General Web Services recommendations [GWS, 04a], [GWS, 04b]. The IMS General Web Services Base Profiles [GWS, 04a] provide a basic structure for the definition of Web Services. They consist of a set of non-proprietary Web services specifications, along with clarifications and amendments to those specifications that promote interoperability. The General Web Services Base Profiles address the most common problems experienced when implementing web service specifications. The IMS Enterprise Service is based upon the General Web Services Base Profile.

The IMS General Web Services UML to WSDL Binding Transformation Rules [GWS, 04b] outlines a process for creating web service bindings using the Base Profiles, Abstract Framework and business domain knowledge intrinsic to the Information Model of a particular Specification. This includes guidelines that instruct project teams in how to use the Work Aids in gathering information, processing information, and specifying a Web Services protocols and binding as appropriate. The methodology includes information and graphics describing the role and relationship of the General Web Services methodology to the IMS Specifications. The IMS Enterprise Services bindings have been created using these transformation rules.

2.3 The Abstract Framework

The IMS Abstract Framework is an approach to enable the IMS to describe the context within which it develops its e-learning technology specifications. This framework is not an attempt to define the IMS architecture, rather it is a mechanism to define the set of interfaces for which IMS may or may not produce a set of interoperability specifications. It is the intention of IMS that the Abstract Framework and the associated IMS specifications produced to realize the exchange of information between the identified services will be adopted in a manner suitable for a particular system requirement. The core features of the framework are:

- Application layer – the set of systems, tools and applications that are constructed from the suite of application and common services to provide a particular set of e-learning functionality;
- Application services layer – the set of entities that provide the e-learning specific services e.g., course management. It is these services that constitute the primary focus for IMS specification development;
- Common services layer – the set of entities that provide the generic services to be used by the application services e.g., authentication;

- Infrastructure layer – the underlying services that enable the exchange of the data structures in terms of physical communications, messaging and corresponding transaction needs;
- Service access points – the access points, or interface, to the corresponding service. Each access point provides access to one service capability;
- Entities – the processes that are used to represent a particular service. The realization of an entity with its service access points is termed a component and its abstract representation is called a Class.

The Enterprise Service and its component services (the Person Management Service, the Group Management Service and the Membership Management Service) are all entities within the Applications Services layer of the IAF. The binding of these services is based upon the usage of a web services infrastructure of SOAPv1.1/HTTPv1.1 defined using WSDLv1.1.

2.4 Related Activities

2.4.1 World Wide Web Consortium (W3C)

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. The IMS makes extensive usage of some of the W3C specifications, in particular:

- XML and related technologies – this is used by IMS as the basis for the data model bindings;
- SOAP – this is used by IMS as one of communication protocols for the service model bindings;
- Web services and WSDL – this is used by IMS as the basis for the service model bindings.

Further information is available at: http://www.w3c.org.

2.4.2 Web Services Interoperability Organization (WS-I)

WS-I is an open, industry organization chartered to promote Web services interoperability across platforms, operating systems, and programming languages. The organization works across the industry and standards organizations to respond to customer needs by providing guidance, best practices, and resources for developing Web services solutions. WS-I was formed specifically for the creation, promotion, or support of Generic Protocols for Interoperable exchange of messages between services. Generic Protocols are protocols that are independent of any specific action indicated by the message beyond actions necessary for the secure, reliable, or efficient delivery of messages; "Interoperable" means suitable for and capable of being implemented in a neutral manner on multiple operating systems and in multiple programming languages.

More details are available at: http://www.ws-i.org.

2.4.3 Open Knowledge Initiative (OKI)

The core deliverable of the Open Knowledge Initiative (OKI) is an architectural specification in support of learning management and educational application development, the primary feature of which is a set of application programming interface (API) definitions. OKI are specifying the architecture by identifying a set of services, a framework, upon which learning tool developers can base their work. See Appendix B for more details.

2.5 Related Specifications

2.5.1 WSDL Specification

The IMS service model bindings are expressed using WSDL v1.1. Version 1.1 is used in preference to version 2.0 due to the maturity of the earlier version and the availability of tools to support the development and adoption of the accompanying WSDL. A WSDL document defines **services** as collections of network endpoints, or **ports**. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: **messages**, which are abstract descriptions of the data being exchanged, and **port types** that are abstract collections of **operations**. The concrete protocol and data format specifications for a particular port type constitute a reusable **binding**. A port is defined by associating a network address with a reusable binding and a collection of ports defines a service.

The IMS Enterprise Services bindings are expressed as WSDL files. The structure of these files is summarized in Section 3.3.

2.5.2 WS-I Profile

The WS-I Basic Profile is shown in Table 2.1.

Table 2.1 WS-I basic profile v1.0.

Core Specification	Description
XML Schema V1.0	All data models will be defined in terms of XML Schema and will require the definition of the corresponding control documents (XSD).
HTTP V1.1 (RFC2616)	HTTP is the mandated protocol binding for the SOAP messages.
SOAP V1.1	SOAP is the mandated messaging protocol.
WSDL V1.1	An instance of the service is defined using WSDL v1.1. WSDL is used to enable the description of services as sets of endpoints operating on messages.
UDDI V2.0	An instance of the service may be defined using a UDDI v2.0 binding template.

It is recognized that SOAP v1.2 and WSDL v1.2 are later versions of the SOAP and WSDL specifications respectively, however, there are no sufficiently robust tools that support these versions. The IMS GWS Basic Profile uses the WS-I basic profile with the exception of the UDDI. See [GWS, 04a] and [GWS, 04b] for further details.

2.5.3 Internet vCard Specification

The vCard specification allows the open exchange of Personal Data Interchange (PDI) information typically found on traditional paper business cards. The specification defines a format for an electronic business card, or vCard. The vCard specification is suitable as an interchange format between applications or systems. The format is defined independent of the particular method used to transport it. The transport for this exchange might be a file system, point-to-point public switched telephone networks, wired-network transport, or some form of unwired transport. The vCard has direct application to the way users utilize the Internet network. The vCard can be used to forward personal data in an electronic mail message. The numerous forms a user of the WWW fills out on a homepage can also be automated using the vCard. The Internet Mail Consortium is working with the Internet Engineering Task Force (IETF) to complete work on an extension to the Internet MIME-based electronic mail standard to allow for this capability. An XML binding of the vCard specification has produced a DTD [vCard, 98] and this has been used to inform the development of the IMS Enterprise Person structure.

The mapping between the IMS Enterprise Service and vCard is shown in Section 6.3.

2.5.4 Internet2 eduPerson

In February 2001, the joint Internet2(R) and EDUCAUSE working group announced the release of the ‘eduPerson’ specification for services that provide seamless access to network-accessible information regardless of where or how the original information is stored. The eduPerson specification provides a set of standard higher-education attributes for an enterprise directory, which facilitate inter-institutional access to applications and resources across the higher education community. The EDUCAUSE/Internet2 eduPerson task force has the mission of defining a Lightweight Directory Access Protocol (LDAP) object class that includes widely-used person attributes in higher education.

The mapping between the IMS Enterprise Service and eduPerson is shown in Section 6.4.

2.5.5 LDAP Specification

The Lightweight Directory Access Protocol (LDAP) is an Internet protocol for accessing distributed directory services that act in accordance with X.500 data and service models. The terms “LDAP” and “LDAPv3” are commonly used to informally refer to the protocol specified by this technical specification. The LDAP suite, as defined here, should be formally identified in other documents by a normative reference to this document. LDAP is an extensible protocol. More detail is available from: http://www.ietf.org.

The mapping between the IMS Enterprise Service and LDAP is shown in Section 6.5.

2.5.6 OKI Open Services Interface Definition (OSID)

Further detail on the OKI OSIDs is given in Appendix B. The mapping between the IMS Enterprise Service and vCard is shown in Section 6.6.

3. Overall Services Model

3.1 Overall Domain Model

A schematic representation of the core objects exchanged using the IMS Enterprise Services specification is given in Figure 3.1; this diagram is based upon the IMS Abstract Framework.

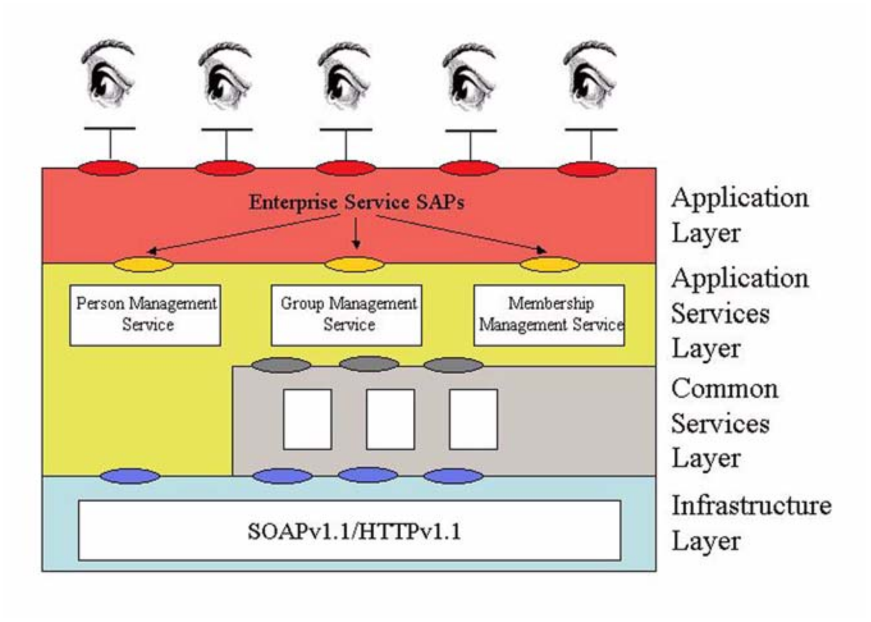


Figure 3.1 Overall Enterprise Service domain model.

The core enterprise services are:

- Person management service – the management of information about individuals who are undertaking some form of study and/or group related activity e.g., they are members of a particular course. The person record is designed to be a data model for all of the personal information to be exchanged about an individual;
- Group management service – the management of information about a collection of objects related to learning activities or individuals. There is no restriction on how the Group and sub-group structures can be used with respect to containing other groups, persons, etc.;
- Membership management service – the management of information about the membership structure is used to define the members of a Group. A member can be a Person or another Group. A Group or Person can be a member of any number of groups. The Group and the member are identified using their sourcedIds.

Two other Enterprise Services have been identified: Gradebook management service and Course catalog management service. They are not be defined as part of the V1.0 information model but may be included in later releases.

3.2 Class Summaries

It is important to note that the UML representation of the interfaces is used to help develop and document the corresponding service information models. It is not a requirement for an implementation to implement this interface as defined i.e., to use the same parameters, etc. Conformance against this specification will be confirmed by inspecting the appropriate binding of the information model and ensuring that the relevant information is present and that different sequences of activity result in the predicted and mandated behavior. It is essential that the behaviors described by each of the operations are fully supported and it is also essential that the behaviors described by different sequences are also maintained.

3.2.1 Person Management Service

The PersonManagementService class is used to model the service responsible for manipulating information about people. The PersonManagementService is split into two interface classes: PersonManager that supports the manipulation of a single Person object; PersonsManager that supports the manipulation of two or more Person objects bound in a single transaction [PMS, 04a]. The manipulation of multiple Person objects can also be supported by the iteration over the PersonManager however this will result in multiple independent transactions. The data-types used to support the PersonsManager class are derived as sets of the equivalent data-types used for the PersonManager class. The service operations are summarized in Table 3.1.

Table 3.1 Summary of PersonManagerService operations.

Operation	Description
createPerson	To request the creation of a populated ‘person’ record on the target system and the source is responsible for the allocation of the unique identifier.
createByProxyPerson	To request the creation of a populated ‘person’ record on the target system and the target is responsible for the allocation of the unique identifier.
deletePerson	To request the deletion of a ‘person’ record. The ‘person’ record is deleted and all of its associated relationships.
readPerson	To read the full contents of the identified ‘person’ record. The target must return all of the data it has for the identified ‘person’ record.
updatePerson	To write new content into the identified ‘person’ record. The target must write the new data into the ‘person’ record. This is an additive operation.
replacePerson	To replace the content of the identified ‘person’ record. The target must write the new data into the ‘person’ record. This is a destructive write-over of all of the original information.
changePersonIdentifier	To change the sourcedId of the ‘person’ record. The completion of this operation will result in later actions using the original sourcedId reporting an unknown identifier status.
createPersons	To request the creation of a set of populated ‘person’ records on the target system The source is responsible for the allocation of the unique identifiers.
createByProxyPersons	To request the creation of two or more populated ‘person’ records on the target system and the target is responsible for the allocation of each the unique identifiers.
deletePersons	To request the deletion of a set of ‘person’ record. The ‘person’ records and all their associated relationships are deleted.
readPersons	To read the full contents of the set of identified ‘person’ records. The target must return all of the data it has for each of the identified ‘person’ records.
readPersonsForGroup	To retrieve the ‘person’ records for a particular Group. This returns the person record for every person that is a member of the Group i.e., for whom a membership record in the Group exists.
updatePersons	To write new content into the set of identified ‘person’ records. The target must write the new data into each of the ‘person’ records. These are additive operations.
replacePersons	To replace the content of a set of identified ‘person’ records. The target must write the new data into the ‘person’ records. These are destructive write-overs of all of the original information.
changePersonsIdentifier	To change the sourcedId of the set of ‘person’ records. The completion of this operation will result in later actions using the original sourcedId reporting an unknown identifier status.

3.2.2 Group Management Service

The GroupManagementService class is used to model the service responsible for manipulating information about groups. The Group Management Service is split into two interface classes: GroupManager that supports the manipulation of a single Group object; GroupsManager that supports the manipulation of two or more Group objects bound in a single transaction [GMS, 04a]. The manipulation of multiple Group objects can also be supported by the iteration over the GroupManager however this will result in multiple independent transactions. The data-types used to support the GroupsManager class are derived as sets of the equivalent data-types used for the GroupManager class. The operations are summarized in Table 3.2.

Table 3.2 Summary of GroupManagerService operations.

Operation	Description
createGroup	To request the creation of a populated 'group' record on the target system and the source is responsible for the allocation of the unique identifier.
createByProxyGroup	To request the creation of a populated 'group' record on the target system and the target is responsible for the allocation of the unique identifier.
deleteGroup	To request the deletion of a 'group' record. The 'group' record is deleted and all of its associated relationships.
deleteGroupRelationship	To request the deletion of a relationship within a 'group' record. This does not delete the associated 'group' records.
readGroup	To read the full contents of the identified 'group' record. The target must return all of the data it has for the identified 'group' record.
updateGroup	To write new content into the identified 'group' record. The target must write the new data into the 'group' record. This is an additive operation.
replaceGroup	To replace the content of the identified 'group' record. The target must write the new data into the 'group' record. This is a destructive write-over of all of the original information.
changeGroupIdentifier	To change the sourcedId of the 'group' record. The completion of this operation will result in later actions using the original sourcedId reporting an unknown identifier status.
createGroups	To request the creation of a set of populated 'group' records on the target system and the source is responsible for the allocation of each of the unique identifiers.
createByProxyGroups	To request the creation of two or more populated 'group' records on the target system and the target is responsible for the allocation of each the unique identifiers.
deleteGroups	To request the deletion of a set of 'group' record. The 'group' records and all their associated relationships are deleted.
deleteGroupsRelationship	To request the deletion of a set of relationships within the 'group' records. This does not delete the associated 'group' records.
readGroups	To read the full contents of the set of identified 'group' records. The target must return all of the data it has for each of the identified 'group' records.
readGroupsForPerson	To retrieve the 'group' records for a particular Person. This returns the set of group records of which the person i.e., for whom a membership record in the Group exists.
updateGroups	To write new content into the set of identified 'group' records. The target must write the new data into each of the 'group' records. These are additive operations.

Operation	Description
replaceGroups	To replace the content of a set of identified ‘group’ records. The target must write the new data into the ‘group’ records. These are destructive write-overs of all of the original information.
changeGroupsIdentifier	To change the sourcedId of the ‘group’ record. The completion of this operation will result in later actions using the original sourcedId reporting an unknown identifier status.

3.2.3 Membership Management Service

The MembershipManagementService class is used to model the service responsible for manipulating information about the membership of people or groups in groups. The Membership Management Service is split into two interface classes: MembershipManager that supports the manipulation of a single Membership object; MembershipsManager that supports the manipulation of two or more Membership objects bound in a single transaction [MMS, 04a]. The manipulation of multiple Membership objects can also be supported by the iteration over the Membership Manager however this will result in multiple independent transactions. The data-types used to support the MembershipsManager class are derived as sets of the equivalent data-types used for the Membership Manager class. The operations are summarized in Table 3.3.

Table 3.3 Summary of MembershipManagerService operations.

Operation	Description
createMembership	To request the creation of a populated ‘membership’ record on the target system and the source is responsible for the allocation of the unique identifier.
createByProxyMembership	To request the creation of a populated ‘membership’ record on the target system and the target is responsible for the allocation of the unique identifier.
deleteMembership	To request the deletion of a ‘membership’ record. The ‘membership’ record is deleted and all of its associated relationships.
readMembership	To read the full contents of the identified ‘membership’ record. The target must return all of the data it has for the identified ‘membership’ record.
updateMembership	To write new content into the identified ‘membership’ record. The target must write the new data into the ‘person’ record. This is an additive operation.
replaceMembership	To replace the content of the identified ‘membership’ record. The target must write the new data into the ‘person’ record. This is a destructive write-over of all of the original information.
changeMembershipIdentifier	To change the sourcedId of the ‘membership’ record. The completion of this operation will result in later actions using the original sourcedId reporting an unknown identifier status.
createMemberships	To request the creation of two or more populated ‘membership’ records on the target system and the source is responsible for the allocation of each the unique identifiers.
createByProxyMemberships	To request the creation of a set of populated ‘membership’ records on the target system and the target is responsible for the allocation of each of the unique identifiers.
deleteMemberships	To request the deletion of a set of ‘membership’ record. The ‘membership’ records and all their associated relationships are deleted.
readMemberships	To read the full contents of the set of identified ‘membership’ records. The target must return all of the data it has for each of the identified ‘membership’ records.

Operation	Description
readMembershipsForPerson	To retrieve all the ‘membership’ records for a particular Person. This returns every ‘membership’ record for the identified ‘person’ record.
readMembershipsForGroup	To retrieve all the ‘membership’ records for a particular Group. This returns every ‘membership’ record for the identified ‘group’ record.
updateMemberships	To write new content into the set of identified ‘membership’ records. The target must write the new data into each of the ‘membership’ records. These are additive operations.
replaceMemberships	To replace the content of a set of identified ‘membership’ records. The target must write the new data into the ‘membership’ records. These are destructive write-overs of all of the original information.
changeMembershipsIdentifier	To change the sourcedId of the set of ‘membership’ records. The completion of this operation will result in later actions using the original sourcedId reporting an unknown identifier status.

3.3 WSDL Bindings

The WSDL bindings have been generated using the methodology documented in [GWS 04a] and [GWS, 04b].

3.3.1 Person Management Services

The composition of the synchronous Person Management Services WSDL binding files is listed below [PMS, 04b]:

- ‘imsPersonManServiceSyncv1p0.wsdl’ – the synchronous service specific WSDL binding file. For the Person Management Service this is based upon SOAP/http. This file imports the abstract definitions using the <wsdl:import> construct;
- ‘imsPersonManAbstractSyncv1p0.wsdl’ – the synchronous abstract message definitions that represent the behavior of the Person Management Service operations. This file imports the message XSD using the <xsd:import> construct;
- ‘imsPersonManSyncv1p0.wsdl’ – the synchronous WSDL binding file. This is the single file equivalent of the combination of the ‘imsPersonManServiceSyncv1p0.wsdl’ and ‘imsPersonManAbstractSyncv1p0.wsdl’ files;
- ‘imsPersonManMessSchemav1p0.xsd’ – the XSD definitions for the synchronous and asynchronous messages. This file imports the Person data model XSD using the <xsd:import> construct;
- ‘imsPersonManDataSchemav1p0.xsd’ – the definition of the Person data model. This is the file that was produced by the equivalent data model binding in Enterprise v1.1;
- ‘imsMessBindSchemav1p0.xsd’ – the XSD binding of the message header parts. This includes the message headers for synchronous, polled and asynchronous message models;
- ‘imsEnterpriseCommonSchemav1p0.xsd’ – the XSD binding of the IMS Enterprise Services common data objects. This file is used by the Person message and data model XSDs as well as the IMS message binding XSD;
- ‘wsiwsdlv1p1.xsd’ – this is the reference XSD for the WSDL definition. This file is the WS-I amended version of the original file from W3C;
- ‘wsisoapv1p1.xsd’ – this is the reference XSD for the SOAP extensions to WSDL. This file is from WS-I.

3.3.2 Group Management Services

The composition of the synchronous and asynchronous Group Management Services WSDL binding files is listed below [GMS, 04b]:

- ‘imsGroupManServiceSyncv1p0.wsdl’ – the synchronous service specific WSDL binding file. For the Group Management Service this is based upon SOAP/http. This file imports the abstract definitions using the <wsdl:import> construct;

- ‘imsGroupManAbstractSyncv1p0.wsdl’ – the synchronous abstract message definitions that represent the behavior of the Group Management Service operations. This file imports the message XSD using the <xsd:import> construct;
- ‘imsGroupManSyncv1p0.wsdl’ – the synchronous WSDL binding file. This is the single file equivalent of the combination of the ‘imsGroupManServiceSyncv1p0.wsdl’ and ‘imsGroupManAbstractSyncv1p0.wsdl’ files;
- ‘imsGroupManMessSchemav1p0.xsd’ – the XSD definitions for the synchronous and asynchronous messages. This file imports the Group data model XSD using the <xsd:import> construct;
- ‘imsGroupManDataSchemav1p0.xsd’ – the definition of the Group data model. This is the file that was produced by the equivalent data model binding in Enterprise v1.1;
- ‘imsMessBindSchemav1p0.xsd’ – the XSD binding of the message header parts. This includes the message headers for synchronous, polled and asynchronous message models;
- ‘imsCommonSchemav1p0.xsd’ – the XSD binding of the IMS common data objects. This file is used by the Group message and data model XSDs as well as the IMS message binding XSD;
- ‘wsiwsdlv1p1.xsd’ – this is the reference XSD for the WSDL definition. This file is the WS-I amended version of the original file from W3C;
- ‘wsisoapv1p1.xsd’ – this is the reference XSD for the SOAP extensions to WSDL. This file is from WS-I.

3.3.3 Membership Management Services

The composition of the synchronous and asynchronous Person Management Services WSDL binding files is listed below [MMS, 04b]:

- ‘imsMemberManServiceSyncv1p0.wsdl’ – the synchronous service specific WSDL binding file. For the Membership Management Service this is based upon SOAP/http. This file imports the abstract definitions using the <wsdl:import> construct;
- ‘imsMemberManAbstractSyncv1p0.wsdl’ – the synchronous abstract message definitions that represent the behavior of the Membership Management Service operations. This file imports the message XSD using the <xsd:import> construct;
- ‘imsMemberManSyncv1p0.wsdl’ – the synchronous WSDL binding file. This is the single file equivalent of the combination of the ‘imsMemberManServiceSyncv1p0.wsdl’ and ‘imsMemberManAbstractSyncv1p0.wsdl’ files;
- ‘imsMemberManMessSchemav1p0.xsd’ – the XSD definitions for the synchronous and asynchronous messages. This file imports the Membership data model XSD using the <xsd:import> construct;
- ‘imsMemberManDataSchemav1p0.xsd’ – the definition of the Membership data model. This is the file that was produced by the equivalent data model binding in Enterprise v1.1;
- ‘imsMessBindSchemav1p0.xsd’ – the XSD binding of the message header parts. This includes the message headers for synchronous, polled and asynchronous message models;
- ‘imsCommonSchemav1p0.xsd’ – the XSD binding of the IMS common data objects. This file is used by the Membership message and data model XSDs as well as the IMS message binding XSD;
- ‘wsiwsdlv1p1.xsd’ – this is the reference XSD for the WSDL definition. This file is the WS-I amended version of the original file from W3C;
- ‘wsisoapv1p1.xsd’ – this is the reference XSD for the SOAP extensions to WSDL. This file is from WS-I.

3.3.4 Namespacing

The name spaces used within the bindings are listed in Table 3.4.

Table 3.4 Namespaces and prefixes used in the binding files.

Namespace	Prefix	Usage
–	“tns:”	The target namespace identifier.
http://www.w3.org/2001/XMLSchema	“xsd:”	The XML schema definition namespace.

Namespace	Prefix	Usage
/enterprise/xsd/imsCommonSchemav1p0	“iaf:”	The IMS Enterprise Service common data model definitions namespace.
/common/xsd/imsMessBindSchemav1p0	“isb:”	The IMS message header binding definitions namespace.
/pms/xsd/imsPersonManDataSchemav1p0	“per:”	The data model namespace for the Person class.
/gms/xsd/imsGroupManDataSchemav1p0	“grp:”	The data model namespace for the Group class.
/mms/xsd/imsMemberManDataSchemav1p0	“mem:”	The data model namespace for the Membership class.
/pms/xsd/imsPersonManMessSchemav1p0	“imspms:”	The IMS Person Management Services message binding definitions namespace.
/gms/xsd/imsGroupManMessSchemav1p0	“imsgms:”	The IMS Group Management Services message binding definitions namespace.
/mms/xsd/imsMemberManMessSchemav1p0	“imsmms:”	The IMS Membership Management Services message binding definitions namespace.
/pms/wsdl/imsPersonManAbstractAsyncReqv1p0	“absp:”	The Person Management Service asynchronous abstract definitions file references.
/pms/wsdl/imsPersonManAbstractAsyncResv1p0	“absp:”	The Person Management Service asynchronous abstract definitions file references.
/gms/wsdl/imsGroupManAbstractAsyncReqv1p0	“absg:”	The Group Management Service asynchronous abstract definitions file references.
/gms/wsdl/imsGroupManAbstractAsyncResv1p0	“absg:”	The Group Management Service asynchronous abstract definitions file references.
/mms/wsdl/imsMemberManAbstractAsyncReqv1p0	“absp:”	The Membership Management Service asynchronous abstract definitions file references.
/mms/wsdl/imsMemberManAbstractAsyncResv1p0	“absp:”	The Membership Management Service asynchronous abstract definitions file references.
wsisoapv1p1	“soap:”	The SOAP references used within the WSDL files.
wsiwsdlv1p1	“wsdl:”	The default WSDL files namespace for WSDL v1.1.

4. SOAP/HTTP Messages for Synchronous Services

4.1 The SOAP/HTTP Message Structures

All of the messages described in the following examples have the same basic structure. The structure of these Request/Response messages is now described.

4.1.1 Request Messages

The basic SOAP/HTTP request message is shown below. The HTTP components are shown in lines (1-5). The invoking service name is shown in line (1) with the host server identified in line (2). The associated SOAP Action is given in line (5). The associated SOAP message structure is supplied in lines (7-9) i.e., enclosed in the element <SOAP-ENV:Envelope>. The usage of the namespace 'SOAP-ENV:' is defined in line (7).

```
0001    POST /PersonManagementService HTTP/1.1
0002    Host: www.personmanagementserver.com
0003    Content-Type: text/xml; charset="utf-8"
0004    Content-Length: nnnn
0005    SOAPAction: "http://www.imsglobal.org/soap/pms/deletePerson"
0006
0007    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0008    …
0009    </SOAP-ENV:Envelope>
```

The detailed structure of the SOAP envelope is shown below. The contents of the SOAP envelope are listed in lines (7-15). The envelope consists of the SOAP header, lines (8-12) and the SOAP body, lines (13-15). The SOAP header is used to contain the message control information that is required for the synchronous end-to-end messaging to support the service behaviors. The header namespace is defined in line (9).

Every request message has a unique message identifier, i.e., line (10). The transmitting system is responsible for creating this unique identifier. The structure for this header is shown in the file 'imsMessBindSchemav1p0.xsd'.

```
0001    POST /PersonManagementService HTTP/1.1
0002    Host: www.personmanagementserver.com
0003    Content-Type: text/xml; charset="utf-8"
0004    Content-Length: nnnn
0005    SOAPAction: "http://www.imsglobal.org/soap/pms/deletePerson"
0006
0007    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0008       <SOAP-ENV:Header>
0009          <h:syncRequestHeaderInfo xmlns:h="../imsMessBindSchemav1p0.xsd">
0010             <h:messageIdentifier>AB12345e4t6789</h:messageIdentifier>
0011          </h:syncRequestHeaderInfo>
0012       </SOAP-ENV:Header>
0013       <SOAP-ENV:Body>
0014       …
0015       </SOAP-ENV:Body>
0016    </SOAP-ENV:Envelope>
```

The SOAP body is shown below in lines (13-19). The namespace to be used in the body is defined in line (14). Line (14) also contains the associated XSD control document file.

```
0001    POST /PersonManagementService HTTP/1.1
0002    Host: www.personmanagementserver.com
0003    Content-Type: text/xml; charset="utf-8"
0004    Content-Length: nnnn
0005    SOAPAction: "http://www.imsglobal.org/soap/pms/deletePerson"
0006
0007    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0008       <SOAP-ENV:Header>
0009          <h:syncRequestHeaderInfo xmlns:h="../imsMessBindSchemav1p0.xsd">
0010             <h:messageIdentifier>AB12345e4t6789</h:messageIdentifier>
0011          </h:syncRequestHeaderInfo>
0012       </SOAP-ENV:Header>
0013       <SOAP-ENV:Body>
0014          <m:deletePersonRequest xmlns:m="../imsPersonManMessSchemav1p0.xsd">
0015             <m:sourcedId>
0016                <esx:identifier>oldsource:oldidentifier</esx:identifier>
0017             </m:sourcedId>
0018          </m:deletePersonRequest>
0019       </SOAP-ENV:Body>
0020    </SOAP-ENV:Envelope>
```

4.1.2 Response Messages

The basic SOAP/HTTP response message is shown below. The associated SOAP message structure is supplied in lines (5-7) i.e., enclosed in the element <SOAP-ENV:Envelope>. The usage of the namespace 'SOAP-ENV:' is defined in line (5).

```
0001    HTTP/1.1 200 OK
0002    Content-Type: text/xml; charset="utf-8"
0003    Content-Length: nnnn
0004
0005    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0006    …
0007    </SOAP-ENV:Envelope>
```

The detailed structure of the SOAP envelope is shown below. The contents of the SOAP envelope are listed in lines (5-19). The envelope consists of the SOAP header, lines (6-15) and the SOAP body, lines (16-18). The SOAP header is used to contain the message control information that is required for the synchronous end-to-end messaging to support the service behaviors. The header namespace is defined in line (7).

Every response message has a unique message identifier, i.e., line (8). The transmitting system is responsible for creating this unique identifier. The structure for this header is shown in the file 'imsMessBindSchemav1p0.xsd'.

Every response message has a status field. The status field for a single transaction it is supplied as <h:statusInfo> whereas for multiple transactions it is supplied as <h:statusInfoSet>. The content of the status information is defined in the IMS Common Data Model [ESCommon, 04].

```
0001    HTTP/1.1 200 OK
0002    Content-Type: text/xml; charset="utf-8"
0003    Content-Length: nnnn
0004
0005    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0006       <SOAP-ENV:Header>
0007          <h:syncResponseHeaderInfo xmlns:h="../imsMessBindSchemav1p0.xsd">
0008             <h:messageIdentifier>AB12345e4t6889</h:messageIdentifier>
0009             <h:statusInfo>
0010                <h:codeMajor>success</h:codeMajor>
0011                <h:severity>status</h:severity>
0012                <h:messageIdRef>AB12345e4t6789</h:messageIdRef>
0013             </h:statusInfo>
0014          </h:syncResponseHeaderInfo>
0015       </SOAP-ENV:Header>
0016       <SOAP-ENV:Body>
0017          …
0018       </SOAP-ENV:Body>
0019    </SOAP-ENV:Envelope>
```

The SOAP body is shown below in lines (16-18). The namespace to be used in the body is defined in line (17). Line (17) also contains the associated XSD control document file.

```
0001    HTTP/1.1 200 OK
0002    Content-Type: text/xml; charset="utf-8"
0003    Content-Length: nnnn
0004
0005    <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0006       <SOAP-ENV:Header>
0007          <h:syncResponseHeaderInfo xmlns:h="../imsMessBindSchemav1p0.xsd">
0008             <h:messageIdentifier>AB12345e4t6889</h:messageIdentifier>
0009             <h:statusInfo>
0010                <h:codeMajor>success</h:codeMajor>
0011                <h:severity>status</h:severity>
0012                <h:messageIdRef>AB12345e4t6789</h:messageIdRef>
0013             </h:statusInfo>
0014          </h:syncResponseHeaderInfo>
0015       </SOAP-ENV:Header>
0016       <SOAP-ENV:Body>
0017          <m:deletePersonResponse xmlns:m="../imsPersonManMessSchemav1p0.xsd"/>
0018       </SOAP-ENV:Body>
0019    </SOAP-ENV:Envelope>
```

4.2 Person Management Service Messages

The set of example Person Management Service SOAP/HTTP messages are described in Table 4.1. These files are stored in the directory '/examples/pms/sync/<operationname>/'. To access the example click on the corresponding filename listed in Table 4.1.

Table 4.1 PMS synchronous binding SOAP/HTTP message examples.

Activity	Request Message Example File	Response Message Example File
Creation of a Single Person Record	createPersonRequest.txt	createPersonResponse.txt
Proxy Creation of a Single Person Record	createByProxyPersonRequest.txt	createByProxyPersonResponse.txt
Deletion of a Single Person Record	deletePersonRequest.txt	deletePersonResponse.txt

Activity	Request Message Example File	Response Message Example File
Reading of a Single Person Record	readPersonRequest.txt	readPersonResponse.txt
Updating of a Single Person Record	updatePersonRequest.txt	updatePersonResponse.txt
Replacing of a Single Person Record	replacePersonRequest.txt	replacePersonResponse.txt
Changing the Identifier of a Single Person Record	changePersonIdentifierRequest.txt	changePersonIdentifierResponse.txt
Creation of Multiple Person Records	createPersonsRequest.txt	createPersonsResponse.txt
Proxy Creation of Multiple Person Records	createByProxyPersonsRequest.txt	createByProxyPersonsResponse.txt
Deletion of Multiple Person Records	deletePersonsRequest.txt	deletePersonsResponse.txt
Reading of Multiple Person Records	readPersonsRequest.txt	readPersonsResponse.txt
Reading of Persons that are a Member of a Group	readPersonsForGroupRequest.txt	readPersonsForGroupResponse.txt
Updating of Multiple Person Records	updatePersonsRequest.txt	updatePersonsResponse.txt
Replacing of Multiple Person Records	replacePersonsRequest.txt	replacePersonsResponse.txt
Changing the Identifier of Multiple Person Records	changePersonsIdentifierRequest.txt	changePersonsIdentifierResponse.txt

4.3 Group Management Service Messages

The set of example Group Management Service SOAP/HTTP messages are described in Table 4.2. These files are stored in the directory '/examples/gms/sync/<operationname>/'. To access the example click on the corresponding filename listed in Table 4.2.

Table 4.2 GMS synchronous binding SOAP/HTTP message examples.

Activity	Request Message Example File	Response Message Example File
Creation of a Single Group Record	createGroupRequest.txt	createGroupResponse.txt
Proxy Creation of a Single Group Record	createByProxyGroupRequest.txt	createByProxyGroupResponse.txt
Deletion of a Single Group Record	deleteGroupRequest.txt	deleteGroupResponse.txt
Deletion of Single Group Relationship	deleteGroupRelationshipRequest.txt	deleteGroupRelationshipResponse.txt
Reading of a Single Group Record	readGroupRequest.txt	readGroupResponse.txt
Updating of a Single Group Record	updateGroupRequest.txt	updateGroupResponse.txt

Activity	Request Message Example File	Response Message Example File
Replacing of a Single Group Record	replaceGroupRequest.txt	replaceGroupResponse.txt
Changing the Identifier of a Single Group Record	changeGroupIdentifierRequest.txt	changeGroupIdentifierResponse.txt
Creation of Multiple Group Records	createGroupsRequest.txt	createGroupsResponse.txt
Proxy Creation of Multiple Group Records	createByProxyGroupsRequest.txt	createByProxyGroupsResponse.txt
Deletion of Multiple Group Records	deleteGroupsRequest.txt	deleteGroupsResponse.txt
Deletion of Multiple Group Relationship	deleteGroupsRelationshipRequest.txt	deleteGroupsRelationshipResponse.txt
Reading of Multiple Group Records	readGroupsRequest.txt	readGroupsResponse.txt
Reading of Groups that have a particular Person as a Member.	readGroupsForPersonRequest.txt	readGroupsForPersonResponse.txt
Updating of Multiple Group Records	updateGroupsRequest.txt	updateGroupsResponse.txt
Replacing of Multiple Group Records	replaceGroupsRequest.txt	replaceGroupsResponse.txt
Changing the Identifier of Multiple Group Records	changeGroupsIdentifierRequest.txt	changeGroupsIdentifierResponse.txt

4.4 Membership Management Service Messages

The set of example Membership Management Service SOAP/HTTP messages are described in Table 4.3. These files are stored in the directory '/examples/mms/sync/<operationname>/'. To access the example click on the corresponding filename listed in Table 4.3.

Table 4.3 MMS synchronous binding SOAP/HTTP message examples.

Activity	Request Message Example File	Response Message Example File
Creation of a Single Membership Record	createMembershipRequest.txt	createMembershipResponse.txt
Proxy Creation of a Single Membership Record	createByProxyMembershipRequest.txt	createByProxyMembershipResponse.txt
Deletion of a Single Membership Record	deleteMembershipRequest.txt	deleteMembershipResponse.txt
Reading of a Single Membership Record	readMembershipRequest.txt	readMembershipResponse.txt
Updating of a Single Membership Record	updateMembershipRequest.txt	updateMembershipResponse.txt
Replacing of a Single Membership Record	replaceMembershipRequest.txt	replaceMembershipResponse.txt

Activity	**Request Message Example File**	**Response Message Example File**
Changing the Identifier of a Single Membership Record	changeMembershipIdentifierRequest.txt	changeMembershipIdentifierResponse.txt
Creation of Multiple Membership Records	createMembershipsRequest.txt	createMembershipsResponse.txt
Proxy Creation of Multiple Membership Records	createByProxyMembershipsRequest.txt	createByProxyMembershipsResponse.txt
Deletion of Multiple Membership Records	deleteMembershipsRequest.txt	deleteMembershipsResponse.txt
Reading of Multiple Membership Records	readMembershipsRequest.txt	readMembershipsResponse.txt
Reading of the Memberships for a Particular Person.	readMembershipsForPersonRequest.txt	readMembershipsForPersonResponse.txt
Reading of the Memberships for a Particular Group.	readMembershipsForGroupRequest.txt	readMembershipsForGroupResponse.txt
Updating of Multiple Membership Records	updateMembershipsRequest.txt	updateMembershipsResponse.txt
Replacing of Multiple Membership Records	replaceMembershipsRequest.txt	replaceMembershipsResponse.txt
Changing the Identifier of Multiple Membership Records	changeMembershipsIdentifierRequest.txt	changeMembershipsIdentifierResponse.txt

5. SOAP/HTTP Messages for Asynchronous Services

Note: At the time of publication the Asynchronous Binding had not been tested. The Asynchronous Binding example details will be issued in V2.0 of this document once testing has been completed.

6. Mapping for Other Specifications

6.1 IMS Enterprise Specification v1.1

6.1.1 Person Management Service

The Person Management Services is the service that is used to exchange the equivalent information contained within the Person data model as described in the IMS Enterprise Information Model v1.1 [Enterprise, 03a]. The support of the v1.1 Person data model by the new Person Management Service is shown in Table 6.1.

Table 6.1 Support for the v1.1 Person data model by the PMS.

IMS Enterprise v1.1 Person Data Model			**IMS Person Management Services Equivalent**		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-recstatus	integer1	O	Unrequired.	–	–
-comments	string2048	O	<recordInfo>	string2048	O
-lang	string128	O	Unsupported	–	–
-sourcedid		M*	Passed as a parameter in the service behavior.	string4096	
-sourcedidtype	string16	O			
-source	string32	M			
-id	string256	M			
-userid	string256	O*	<userId><userIdValue>	string256	O
-useridtype	string32	O	<userId><userIdType>	string32	O
-password	string1024	O	<userId><passWord>	string1024	O
-pwencryptiontype	string32	O	<userId><pwEncryptionType>	string32	O
-authenticationtype	string32	O	<userId><authenticationType>	string32	O
-name		M	See below	–	–
-fn	string256	M	<formatName>	string256	O
-sort	string256	O	<name> <partname> <namePartType> <namePartValue>	string32 string256	O
-nickname	string256	O	<name> <partname> <namePartType> <namePartValue>	string32 string256	O
-n		O	Unrequired	–	–
-family	string256	O	<name> <partname> <namePartType> <namePartValue>	string32 string256	O
-given	string256	O	<name> <partname> <namePartType> <namePartValue>	string32 string256	O

IMS Enterprise v1.1 Person Data Model			IMS Person Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-other	string256	O*	<name> <partname> <namePartType> <namePartValue>	 string32 string256	O
-prefix	string32	O	<name> <partname> <namePartType> <namePartValue>	 string32 string256	O
-suffix	string32	O	<name> <partname> <namePartType> <namePartValue>	 string32 string256	O
-partname	string256	O*	<name> <partname> <namePartType> <namePartValue>	 string32 string256	O
-lang	string128	O			
-partnametype		M			
-demographics		O	<demographics>	–	O
-gender	string1	O	<gender>	enumerated	O
-bday	datetime	O	<bday>	date	O
-disability	string64	O	<disability>	string32	O
-email	string256	O	<email>	string2048	O
-url	url	O	<url>	string4096	O
-tel	string32	O4	<tel><telValue>	string32	M
-teltype	string8	O	<tel><telType>	enumerated	O
-adr			<address>	–	O
-pobox	string32	O	<pobox>	string32	O
-extadd	string128	O	<extadd>	string128	O
-street	string128	O3	<street>	string128	O3***
-locality	string64	O	<locality>	string64	O
-region	string64	O	<region>	string64	O
-pcode	string32	O	<postcode>	string32	O
-country	string64	O	<country>	string64	O
-photo			<photo>		O
-extref	string1024	M	<extref>	string1024	M
-imgtype	string32	O	<imgType>	string32	O
-systemrole		O	<systemRole>	enumerated	O
-systemroletype	string32	M	Unrequired	–	–
-institutionrole		O*	<institutionRole>	–	O*
-primaryrole	string4	M	<primaryRoleType>	boolean	M

IMS Enterprise v1.1 Person Data Model			IMS Person Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-institutionroletype	string32	M	<institutionRoleType>	enumerated	M
-datasource	string256	O	<dataSource>	string2048	O

6.1.2 Group Management Service

The Group Management Services is the service that is used to exchange the equivalent information contained within the Group data model as described in the IMS Enterprise Information Model v1.1 [Enterprise, 03a]. The support of the v1.1 Group data model by the new Group Management Service is shown in Table 6.2.

Table 6.2 Support for the v1.1 Group data model by the GMS*.**

IMS Enterprise v1.1 Group Data Model			IMS Group Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-recstatus	integer1	O	Unrequired.	–	–
-comments	string2048	O	<recordInfo>	string2048	O
-lang	string128	O	Unsupported	–	–
-sourcedid		M*	Passed as a parameter in the service behavior.	string4096	
-sourcedidtype	string16	O			
-source	string32	M			
-id	string256	M			
-grouptype		O*	<groupType>		M
-scheme	string256	O	<scheme>	string256	M
-typevalue	string256	M*	<typeValue> <type>	string256	M* M
-level	string2	M	<level>	string2	M
-description		M	<description>		O
-short	string60	M	<descShort>	string64	M
-long	string256	O	<descLong>	string256	O
-full	string2048	O	<descFull>	string2048	O
-org		O	<org>		O
-orgname	string256	M	<orgName>	string256	O
-orgunit	string256	O*	<orgUnit>	string256	O*
-type	string32	O	<orgType>	string32	O
-id	string256	O	<id>	string256	O
-timeframe		O	<timeFrame>		O
-begin	datetime	O	<begin> <date>	date	O M
-restrict	integer1	M	<restrict>	boolean	M
-end	datetime	O	<end> <date>	date	O M
-restrict	integer1	M	<restrict>	boolean	M
-adminperiod	string32	O	<adminPeriod>	string32	O

IMS Enterprise v1.1 Group Data Model			IMS Group Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-enrollcontrol		O	<enrollControl>		O
-enrollaccept	integer1	O	<enrollAccept>	boolean	O
-enrollallowed	integer1	O	<enrollAllowed>	boolean	O
-email	string256	O	<email>	string2048	O
-url	url	O	<url>	string4096	O
-relationship		O	<relationship>		O*
-relation	string8	O	<relation>	string8	M
-sourcedid		M	<sourcedId><identifier>	string4096	M
-sourcedidtype	string16	O			
-source	string32	M			
-id	string256	M			
-label	string32	M	<label>	string32	M
-datasource	string256	O	<dataSource>	string2048	O

6.1.3 Membership Management Service

The Membership Management Services is the service that is used to exchange the equivalent information contained within the Membership data model as described in the IMS Enterprise Information Model v1.1 [Enterprise, 03a]. The support of the v1.1 Membership data model by the new Membership Management Service is shown in Table 6.3.

Table 6.3 Support for the v1.1 Membership data model by the MMS.

IMS Enterprise v1.1 Membership Data Model			IMS Membership Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-comments	string2048	O	<recordInfo>	string2048	O
-lang	string128	O	Unsupported	–	–
-sourcedid		M	<groupSourcedId> <identifier>	string4096	M
-sourcedidtype	string16	O			
-source	string32	M			
-id	string256	M			
-member		M*	<member>		M
-comments	string2048	O	<recordInfo>	string2048	O
-lang	String128	O	Unsupported	–	–
-sourcedid		M	<memberSourcedId> <identifier>	string4096	M
-sourcedidtype	string16	O			
-source	string32	M			
-id	string256	M			
-idtype	integer1	M	<idType>	string	O
-role		M*	<role>		M*

IMS Enterprise v1.1 Membership Data Model			IMS Membership Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-recstatus	integer1	O	Unrequired	–	–
-roletype	string32	M	<roleType>	string32	M
-lang	string128	O	Unsupported	–	–
-subrole	string32	O	<subRole>	string32	O
-status	integer1	M	<status>	boolean	O
-userid	string256	O*	<userId> <userIdValue>	string256	O
-useridtype	string32	O	<userId> <userIdType>	string32	O
-password	string1024	O	<userId> <passWord>	string1024	O
-pwencryptiontyp	string32	O	<userId> <pwEncryptionType>	string32	O
-authenticationty	string32	O	<userId> <authenticationType>	string32	O
-comments	string2048	O	<recordInfo>	string2048	O
-lang	string128	O	Unsupported	–	–
-datetime	datetime	O	<dateTime>	dateTime	
-timeframe		O	<timeFrame>		O
-begin	datetime	O	<begin> <date>	date	O M
-restrict	integer1	M	<restrict>	boolean	M
-end	datetime	O	<end> <date>	date	O M
-restrict	integer1	M	<restrict>	boolean	M
-adminperiod	string32	O	<adminPeriod>	string32	O
-interimresult		O*	<interimResult>		O*
-resulttype	string32	O	<resultType>	string32	O
-mode	string32	O	<mode>	string32	O
-values		O	<values>		O
-valuetype	integer1	M	<valueType>	string	O
-list	string32	O*	<list>	string32	O*
-min	decimal8p4	O	<min>	decimal	O
-max	decimal8p4	O	<max>	decimal	O
-result	string32	O	<result>		
-comments	string2048	O	<recordInfo>	string2048	O
-lang	string128	O	Unsupported	–	–
-finalresult		O*	<finalResult>		O*
-mode	string32	O	<resultType>***	string32	O
-values		O	<mode>	string32	O
-valuetype	integer1	M	<values>		O

IMS Enterprise v1.1 Membership Data Model			IMS Membership Management Services Equivalent		
attribute	**type**	**mult**	**attribute**	**type**	**mult**
-list	string32	O*	<valueType>	string	O
-min	decimal8p4	O	<list>	string32	O*
-max	decimal8p4	O	<min>	decimal	O
-result	string32	O	<max>	decimal	O
-comments	string2048	O	<recordInfo>	string2048	O
-lang	string128	O	Unsupported	–	–
-email	string256	O	<email>	string2048	O
-datasource	string256	O	<dataSource>	string2048	O

6.2 IMS Learner Information Package (LIP)

The mapping between the IMS Person Management Services <person> element and the IMS LIP <identification> element is shown in Table 6.4.

Table 6.4 Mapping between PMS <person> and IMS LIP <identification>.

Original <person> sub-elements	Equivalent <identification> sub-elements
<sourcedId>SourceEntry1:IdEntry2	<contentype>
	<referential>
	<sourcedid>
	<source>SourceEntry1
	<id>IdEntry2
<userId>	<ext_identification>
<userIdValue>Entry1	<extensionelement1>Entry1
<userIdType>Entry2	<extensionelement1>Entry2
<passWord>Entry3	<extensionelement1>Entry3
<pwEncryptionType>Entry4	<extensionelement1>Entry4
<authenticationType>Entry5	<extensionelement1>Entry5
<formatName>Entry	<formname>
	<typename>
	<tyvalue>Full
	<text>Entry
<recordInfo>	<ext_identification>
<comments>Entry	
<name>	<name>
<partName>	<partname>
<partNameType>Entry1	<typename>
<partNameValue>Entry2	<tyvalue>Entry1
	<text>Entry2
<demographics><gender>1	<demographics>
	<gender gender="M"/>
<demographics><gender>2	<demographics>
	<gender gender="F"/>

Original <person> sub-elements	**Equivalent <identification> sub-elements**

<demographics><bday>Entry	<demographics>
	    <date>
	        <typename>
	            <tyvalue>DoB
	        <datetime>Entry

<demographics><disability>Entry	<ext_identification>Entry
<email>Entry	<contactinfo> <email>Entry
<url>Entry	<contactinfo> <web>Entry

<tel>
    <telType>1
    <telValue>Entry
	<contactinfo>
	    <telephone>
	        <areacode>Entry
	        <indnumber>Entry

<tel>
    <telType>2
    <telValue>Entry
	<contactinfo>
	    <facsimile>
	        <areacode>Entry
	        <indnumber>Entry

<tel>
    <telType>3
    <telValue>Entry
	<contactinfo>
	    <mobile>
	        <areacode>Entry
	        <indnumber>Entry

<tel>
    <telType>4
    <telValue>Entry
	<contactinfo>
	    <pager>
	        <areacode>Entry
	        <indnumber>Entry

<address><pobox>Entry	<contactinfo>
	    <address>
	        <pobox>Entry

<address><extadd>Entry	<contactinfo>
	    <address>
	        <street>
	            <nonfieldedstreetaddress>Entry

<address><street>Entry	<contactinfo>
	    <address>
	        <street>
	            <nonfieldedstreetaddress>Entry

<address><locality>Entry	<contactinfo>
	    <address>
	        <locality>Entry

<address><region>Entry	<contactinfo>
	    <address>
	        <region>Entry

<address><pcode>Entry	<contactinfo>
	    <address>
	        <postcode>Entry

<address><country>Entry	<contactinfo>
	    <address>
	        <country>Entry

Original <person> sub-elements	Equivalent <identification> sub-elements

<photo><extref>Entry	<demographics>
	    <representation>
	        <description>
	            <full>
	                <media>Entry

<system_role>Entry	<ext_identification>Entry

<institution_role>
    <institutionRoleType>Entry1
    <primaryRoleType>Entry2
	<ext_identification>Entry1, Entry2

<datasource>Entry	<contentype>
	    <referential>
	        <indexid>Entry

<extension>Entry	<ext_identification>Entry

6.3 IETF vCard Support

The IMS Enterprise Service is compatible with the IETF vCard specification i.e., many of the vCard fields can be contained by an Enterprise-XML instance and the rest are supported through the use of the Person extension element. This relationship is shown in Table 6.5, namely:

Table 6.5 Usage of IMS Person Management Service to support the IETF vCard specification.

vCard Element	IMS PMS Element(s)	Notes
FN	<person><formatName>	The formatted name.

n	<person><name>
	    <partName>
	        <namePartType>
	        <namePartValue>
		The name.

family	<person><name>
	    <partName>
	        <namePartType>Family
	        <namePartValue>
		Family name component.

given	<person><name>
	    <partName>
	        <namePartType>Given
	        <namePartValue>
		Given name component.

other	<person><name>
	    <partName>
	        <namePartType>Other
	        <namePartValue>
		Other name components.

prefix	<person><name>
	    <partName>
	        <namePartType>Prefix
	        <namePartValue>
		Prefix name component.

suffix	<person><name>
	    <partName>
	        <namePartType>Suffix
	        <namePartValue>
		Suffix name component.

vCard Element	IMS PMS Element(s)	Notes

nickname	<person><name>
	  <partName>
	    <namePartType>Nickname
	    <namePartValue>
		Nickname.

photo	<person><photo>	A photograph of the Person.
bday	<person><demographics><bday>	The birth date of the Person.
addr	<person><address>	The address.
pobox	<person><address><pobox>	The PO Box address component.
extadd	<person><address><extadd>	The extended address.
street	<person><address><street>	The street address component.
locality	<person><address><locality>	The locality address component.
region	<person><address><region>	The region address component.
pcode	<person><address><postcode>	The post code/zip code address component.
country	<person><address><country>	The country address component.
label	<person><extension>	Requires the usage of the Person extension feature.
tel	<person><tel>	The telephone number.
email	<person><email>	The email address.
mailer	<person><extension>	Requires the usage of the Person extension feature.
tz	<person><extension>	Requires the usage of the Person extension feature.
geo	<person><extension>	Requires the usage of the Person extension feature.
lat	<person><extension>	Requires the usage of the Person extension feature.
lon	<person><extension>	Requires the usage of the Person extension feature.
title	<person><extension>	Requires the usage of the Person extension feature.
role	<person><extension>	Requires the usage of the Person extension feature.
logo	<person><extension>	Requires the usage of the Person extension feature.
agent	<person><extension>	Requires the usage of the Person extension feature.
org	<person><extension>	Requires the usage of the Person extension feature.
categories	<person><extension>	Requires the usage of the Person extension feature.
item	<person><extension>	Requires the usage of the Person extension feature.
note	<person><extension>	Requires the usage of the Person extension feature.

sort	<person><name>
	  <partName>
	    <namePartType>Sort
	    <namePartValue>
		The sort form for the name.

sound	<person><extension>	Requires the usage of the Person extension feature.
url	<person><url>	The web URL.
key	<person><userId>	Security keys.

6.4 Internet2/Educause ‘eduPerson’ Support

The eduPerson specification is an object class for LDAP services whereas Enterprise is a set of data objects for the exchange of learner Enterprise information and not just directory-related information. The relationship between the eduPerson V1.0 specification and the IMS Person management Services is summarized in Table 6.6.

Table 6.6 Usage of IMS Enterprise Service to exchange the eduPerson information

EduPerson Object Definition	IMS PMS Data Structure	Comments
EduPersonAffiliation (OID: 1.3.6.1.4.1.5923.1.1.1.1)	<person><institutionRole>	Specifies the person's relationship(s) to the institution in broad categories such as student, faculty, staff, alum, etc. This is to use a controlled vocabulary and IMS will work with Internet2/Educause to achieve a common vocabulary base.
EduPersonNickname (OID: 1.3.6.1.4.1.5923.1.1.1.2)	<person><name> <partName> <namePartType>Nickname <namePartValue>	Person's nickname, or the informal name by which they are accustomed to be hailed.
EduPersonOrgDN (OID: 1.3.6.1.4.1.5923.1.1.1.3)	<person><extension>	The distinguished name (DN) of the directory entry representing the institution with which the person is associated. The Person extension structure must be used.
EduPersonOrgUnitDN (OID: 1.3.6.1.4.1.5923.1.1.1.4)	<person><extension>	The distinguished name (DN) of the directory entries representing the person's Organizational Unit(s). With a distinguished name, the client can do an efficient lookup in the institution's directory for information about the person's organizational unit(s). The Person extension structure must be used.
EduPersonPrimaryAffiliation (OID: 1.3.6.1.4.1.5923.1.1.1.5)	<person><institutionRole>	Specifies the person's PRIMARY relationship to the institution in broad categories such as student, faculty, staff, alum, etc. This is to use a controlled vocabulary and IMS will work with Internet2/Educause to achieve a common vocabulary base.
EduPersonPrincipalName (OID: 1.3.6.1.4.1.5923.1.1.1.6)	<person><userId>	The "NetID" of the person for the purposes of inter-institutional authentication. Should be stored in the form of user@univ.edu, where univ.edu is the name of the local security domain. This information can be contained within Person <userid> element.

6.5 LDAP Attribute Mapping

Most enterprise systems utilize a directory to store organizational and person information. Many directories use the Lightweight Directory Access Protocol (LDAP) to store this data and make it accessible to other Enterprise applications. It is likely that Learning Management Systems will use some of the data in an LDAP directory to populate equivalent fields in the IMS Enterprise XML binding. Table 6.7 represents a preliminary mapping between LDAP base schema items and IMS Enterprise elements. This is provided only as an example. The reader who wishes to incorporate LDAP data into learning management applications is encouraged to consult authoritative sources regarding LDAP.

Table 6.7 Mapping of IMS Enterprise Services to LDAP attributes.

IMS Data Object	IMS PMS Data Structure	LDAP Attribute Name, Alias
Person	<sourcedId>	Uid

	<name>
	  <partName>
	    <namePartType>Common
	    <namePartValue>
		cn, commonName

	<name>
	  <partName>
	    <namePartType>Surname
	    <namePartValue>
		sn, surName

	<name>
	  <partName>
	    <namePartType>Given
	    <namePartValue>
		givenName

	<tel>	telephoneNumber
	<address>	postalAddress
	<address><pobox>	postOfficeBox
	<address><street>	street
	<address><locality>	l, locality, localityName
	<address><country>	c, countryName
	<photo>	photo
Group	<sourcedId>	o, organization
	<org><orgName>	o, organization
	<org><orgUnit>	ou, organizationUnitName
	<org><orgType>	businessCategory
Membership	<member><role><userId>	Uid

6.6 OKI Enterprise Service OSIDs

The OKI definition of ‘Group’ is different to that by IMS, but there are still ways to relate the concepts found in both. The idea of a ‘Group’ in OKI is a set of people, whose purpose is left undefined and who are undifferentiated. A ‘Group’ can have subgroups as well as members, but the membership in a group is undifferentiated i.e., you are either a member or not.

The concept of ‘Membership’ in IMS is that a person has one or more roletypes within a group. In OKI this would be expressed using the Authorization SID, by mapping IMS roletypes as functions and treating IMS groups as qualifiers.

These are really two different uses of group. In one sense it is a just a set of people and in another it is just a placeholder for some other entity such as a course. In OKI a ‘Group’ represents a set of people and what this means is left to the application. Therefore, a group could be constructed to represent the set of students in a course, but it wouldn’t represent the course itself. In an application you would associate the group to the course. In this case the Course would be represented as a Qualifier in (OKI Authorization OSID). The Qualifier Identifier in the Authorization Service would be set to the CourseIdentifier. You would use the Authorization OSID to maintain and query peoples roles associated with the course. Groups used for mailing lists etc. could be created using the OKI Shared OSID, if this is also needed.

The IMS Membership structure also contains results information. Results information in OKI is associated with a Course and a Person, and the methods found in the CourseManagement OSID.

In OKI there are different levels of abstractions, some more general and some more specific. In the Enterprise domain, OKI has three constructs for relationships:

- Groups – sets of people;
- Authorizations – assertions about people and their functions (or roles) within some context;
- CourseOfferings – represent courses and their information.

Therefore, depending on what your using IMS Group and membership for, you may want to relate it to OKI in different ways.

Further detail on the relevant OKI OSIDs is given in Appendix B.

7. Best Practice

7.1 Achieving Interoperability

The following list describes some types of systems for which the IMS Enterprise Services specification may support Learning Management interoperability. It includes a list of some interfaces that the current specification can support.

7.1.1 Human Resource Management System

Human Resource Management Systems (HRMS) manage personnel records, payroll, benefits, competency management, and other functions for an enterprise. Interoperability that can be supported by this specification include:

From HRMS to LMS:

- Person data maintained in the HRMS and passed to the LMS;
- HR departments passed as groups, and employees of those departments passed as members;
- Special groups of employees (new hires for example) passed to the LMS as training groups.

From LMS to HRMS:

- After the completion of training courses, course information is returned to the HRMS as groups, and completion of training courses, or it could come back as membership in those groups, with result information included.

7.1.2 Corporate Training Management System

Corporate Training Administration systems keep track of employee training plans, schedule training courses (including instructors and resources), enroll people in training, record training completed, and update employee competencies in an HRMS. They are also used to manage training delivered to customers. Interoperability that can be supported by this specification include:

From Training to LMS:

- Person data might be passed to the LMS from the training system;
- Training groups (courses) and memberships could be passed from training to the LMS.

From LMS to Training:

- After the completion of training courses, membership objects could be sent to the Training Administration system from the LMS with result (completion) information included.

7.1.3 Student Administration System

Student Administration Systems (SAS) keep track of student education plans, schedule courses (including instructors and resources), enroll people in courses, record course results, and update student academic progress. Interoperability that can be supported by this specification include:

From SAS to LMS:

- Person data for people enrolled in groups that are managed by the LMS;
- Group data could be passed from SAS to the LMS to create the groups;
- Group membership (course enrollment) data may be passed from SAS to the LMS;
- Final grade information may be passed to the LMS from the SAS in an updated Membership object if final grading occurs in the SAS, and the LMS needs the final grade for its records.

From LMS to SAS:

- Final grades could be returned to SAS from the LMS by passing back Membership records with the Result data provided. This data could then be entered into a formal grade roster process on the SA side.

7.1.4 Library Management System

Library Management systems can be thought of as a particular class of Learning Management system, in that they provide a set of services for managing the interaction of learners with learning objects. Therefore, it is appropriate to use this specification to support interfaces from other enterprise systems to Library Management systems in much the same way that these interfaces are supported with Learning Management Systems.

From SAS or HRMS to Library:

- People data;
- Groups – course sections for access to specific material, HR departments for access to services, alumni for access to limited services, etc.;
- Group membership.

7.2 Implementing the Abstract API

7.2.1 Single Transaction/Single Operation

The three services have operations that allow individual data objects to be manipulated. Each of these operations, contained within the ‘PersonManager’, ‘GroupManager’ and ‘MembershipManager’ interface classes, results in a single request/response message exchange. Each operation manipulates the state of one object (in some cases there may be ripple effects to ensure consistency across the full data set) and reports the result of that action. Therefore, these operations support a single transaction.

7.2.2 Multiple Transactions/Multiple Operations

If multiple transactions are required then this can be achieved by iterating across the single operations i.e., if five new person objects are to be created then five create operations can be issued sequentially. Each operation will carry a single transaction and so five operation calls results in five individual response/message exchanges. The advantage of this approach is that no new implementation features are required and there is an incremental change of state and it’s reporting. The disadvantage is that for a large number of similar operations there is a significant communication overhead that could result in the communications network becoming overloaded. At the very least there will be a significant communications delay before all of the transactions are completed.

7.2.3 Multiple Transactions/Single Operation

An alternative approach for multiple transactions is to use the operations contained within the ‘PersonsManager’, ‘GroupsManager’ and ‘MembershipsManager’ interface classes. The corresponding operations enable many transactions on the same category of objects to be invoked in a single operation. In the above example of the creation of five new person objects, this would result in a single operation call containing the five bundled transactions. This results in a single request/response message. The advantage of this approach is that there is no extra communication overhead i.e., there is a single request/response message interaction. The disadvantage is that for a synchronous service there is no response until all of the transactions have been completed – this could be a long delay. The response reports on the status of each transaction and so the service requester must process the received message to pair the status information in the SOAP header with the corresponding data in the SOAP body. The SOAP messages contain all of the relevant linkage information.

7.2.4 Single and Multiple Sessions

The concept of a session is outside the scope of the specification. New operations can be defined to introduce the concept of a session but for asynchronous bindings this will need to address the implications of multiple sessions.

7.2.5 Identifiers and SourcedIds

In the IMS Enterprise v1.1 specification the ‘sourcedId’ was a structured object i.e., it consisted of ‘source’ and ‘id’ sub-structures. In the IMS Enterprise Services specification the ‘sourcedId’ is based upon the ‘identifier’ structure that is defined as a flat string. This flat string can be used to contain the sub-structured format from Enterprise v1.1 specification using the following algorithm:

IMS Enterprise Services ‘sourcedId’ = <source>&…&<id>

Where:

<source>	is the value of the source element in the IMS Enterprise v1.1 specification implementation;
<id>	is the value of the id element in the IMS Enterprise v1.1 specification implementation;
‘&…&’	is the delimiter between the ‘source’ and ‘id’ values. The number of ‘&’ in the sequence must be one greater than the number of concatenated occurrences elsewhere i.e., in either the ‘source’ or the ‘id’ values.

In the case where the ‘source’ = IMS and ‘id’ = wehu12kio then the new ‘sourcedId’ = IMS&wehu12kio. In the case where the ‘source’ = IM&S and ‘id’ = wehu1&&2kio then the new ‘sourcedId’ = IM&S&&&wehu1&&2kio.

7.2.6 Passing More Parameters

The information models define what parameters are to be passed within the SOAP message body. At the current time there is **no** way to extend this set of parameters. If more parameters need to be passed then the following approaches can be considered:

- New operations are defined with similar functionality to those who parameters must be extended. The new definitions will include the new parameters;
- New operations are defined that establish an end-to-end session. The parameters passed in these session-establishing behaviors would then have meaning throughout the session.

IMS welcomes feedback on the issue of adding new parameters and how to best facilitate this in the specification.

7.2.7 Creating an Implementation API

The IMS Enterprise Services specification defines an abstract API. This API is defined to enable the corresponding request/response to be created and represented in WSDL. There is no requirement to directly convert the abstract API to a language dependent implementation equivalent i.e., a Java API does not have to provide the ‘createPerson’ method, etc.

It is recommended that an appropriate implementation API be created to insulate the rest of the application from the communications handler responsible for Enterprise systems interoperability. This API should take the form most appropriate to the business process being supported by the Enterprise Services specification. This API will then provide the adaptation between those business processes and the creation and handling of the SOAP messages that are defined within the Enterprise Services binding.

The implementation API could also support other operations that have not been defined with the IMS Enterprise Services specification. This is one way in which the IMS specification can be extended. The only constraint is that the same message structure and choreography is followed. This ensures that any Service Provider can reject an unknown service request by returning the stats code ‘unsupported’ in the SOAP message header. Conversely, every implementation must be capable of rejecting unknown service requests. Any subsequent local error message logging etc. is implementation dependent.

7.3 Status Codes and SOAP Fault Messages

7.3.1 Status Codes

The IMS Enterprise Services documentation and the IMS Common Data Definitions documents give an extension description of the set of status codes that can be reported and the conditions under which the codes are to be reported. There are two further issues to be considered:

- Request authority – if the Service Requester does not have the appropriate authority for the issued request then the Service Provider will reject the request issuing the appropriate status code. The default codeMinor value is: ‘authorizationfail’;

- Conformance level mismatch – if there is a mismatch between the conformance levels of the two systems then there will be data exchange problems. In these cases the systems must exchange the maximum amount of data from their perspective. If the Service Provider cannot store all of the data it has been given then it returns a codeMajor/severity value of 'Success/Warning' with a codeMinor value of 'partialdatastorage'. If the Service Provider supplies too much information for the Service Requester then the invoking application receive the report of codeMajor/severity value of 'Success/Warning' with a codeMinor value of 'receivedataoverload'

7.3.2 SOAP Fault Codes

The SOAP faults are reported in the SOAP header. This means that when a SOAP request message is issued the response message may contain SOAP fault codes with no further useful information. It is the responsibility of the implementations of the Service Requester and Service Provider to convert the SOAP fault codes to the equivalent IMS Enterprise Services status codes. The default codeMajor/severity values are 'Failure/Error' and the codeMinor value is 'soapfault'. More detailed codeMinor codes can be created if required.

7.3.3 Handling the Status Codes

The IMS Enterprise Services specification does not describe how the status codes are to be passed from the Service Requester and Service Provider communications handlers i.e., the usage of the 'StatusInfo' and 'StatusInfoSet' objects is a part of the abstract API. An implementation API must describe how the status information is to be passed t the driving applications. There are several alternatives including being passed as a parameter in the API interface and requiring interrogation using a special status code API interface call. Clearly, the manner in which the status codes are handled in the Service Requester and Service Provider do not have to be the same. The only requirement is that all of the status information must be passed in the SOAP message header. It is also required that the SOAP fault codes will also be passed in the same manner as the IMS Enterprise Services status code information.

7.4 Architectural Considerations

7.4.1 Information Synchronization

The IMS Enterprise Service bindings provide mechanisms through which the synchronization of data transfers can be maintained. These mechanisms are:

- Synchronous and asynchronous communications – two different message choreographies are available. The synchronous binding requires the Service Requester to wait for the response from the Service Provider whereas the asynchronous binding allows many service requests to be issued and outstanding;
- Message identifiers – all of the SOAP messages have unique message identifiers. The status information in the response message includes the message identifier of the original request message;
- Sourced identifiers – every data object is allocated a unique identifier. This identifier must be unique in the context of the two systems that access the object i.e., the identifiers do not have to be globally unique. The end systems are responsible for maintaining the integrity of these identifiers.

7.4.2 Push and Pull Transactions

The Enterprise Services are defined in such a way that any system can be either a Service Requester or Service provider or both. Data can be pushed or pulled depending on how the IMS Enterprise Services are used. Pushed data requires the source to issue 'create', createByProxy', 'delete', 'write' and 'replace' operations. Pulled data requires the source to issue 'read' operations.

7.4.3 'Snapshot' and Event Driven Transactions

The IMS Enterprise team's consensus is that the most robust and easily implemented interface would involve the passing of a complete 'snapshot' of the Person, Group, and Group Membership data. The target system would examine this snapshot to determine what changes had occurred. This very basic type of interface allows a receiving system to pick up an interface at any time and synchronize its data with the source system – regardless of how many interfaces had been passed in the interim. A purely event driven ‘transactional’ interface, on the other hand, cannot tolerate any loss or skipping of interface records. The snapshot approach requires the usage of the iterated operations in the three services.

This basic interface also allows the target system to implement many different strategies for dealing with the interface data. Taking a “snapshot” means that the full set of relevant data from the source system can be moved to the target system environment on any timing needed to support the business processes. This interface architecture has the advantage of being very tolerant of lost messages or missed data objects because the next transmittal will always get the target system back in synchronization with the source system. However, the major drawback is that the target system can never be sure that the data has not changed in the source system since the last snapshot was received. Also, this interface architecture does not effectively support two-way interfaces. In a two-way interface, data object maintenance occurs in both systems, and the data objects are passed in both directions.

In an event driven interface, the source system publishes data object messages when events occur. This changes the relevant data, and the target system receives and processes the event transactions. The existence of an event driven interface does not eliminate the usefulness of the “snapshot” interface. Because an event driven interface is not tolerant of missed transactions, the “snapshot” interface can be used at regular intervals to “re-synchronize” the data in the target system with that in the source system. This increases the fault tolerance of the overall interface architecture.

7.4.4 Common Services and Service Choreography

The interaction of the services with each other and with common services is outside the scope of this specification. However, there are recommendations on how these interactions should be managed in the context of the Enterprise Services:

- Authorization, authentication and other similar information should be passed in the SOAP headers. The SOAP headers can be extended using the W3C recommendations;
- The WS-I Basic Profile 1.0 defines the usage of UDDI 2.0 for service discovery etc. At the current time there is little experience in the usage of UDDI but this approach should be considered in any implementation.

7.5 Synchronous and Asynchronous Communications

The information models are created agnostic of the communications infrastructure choreography. Synchronous and asynchronous bindings of the IMS Enterprise Services have been developed and the key differences from an implementation perspective are:

- The synchronous binding has the simple request/response message choreography whereas the asynchronous binding has request/acknowledge and response/acknowledge message exchanges. The co-ordination between the two message sets is implementation dependent;
- For the synchronous binding the service requester is blocked until the response message has been received. It is still important to verify that the response message is correctly matched to the request message (use the ‘messageIdentifier’ and ‘messageRefIdentifier’ structures in the SOAP message headers) because the underlying communications infrastructure may result in unexpected behavior. In the asynchronous binding the service requester is only blocked until the initial acknowledgement message is received from the service provider;
- For the asynchronous binding the service requester must either poll the communications handler for data reception or the communications handler must announce the arrival of data for the service requester. The mechanism adopted is implementation dependent.

For both the synchronous and asynchronous bindings it is assumed that the end-to-end communications is error-free, there is no message re-ordering and no message duplication (the correct usage of the message identifiers in the SOAP headers will protect against some of these problems). In the binding there is no provision of reliable messaging but this will be investigated for adoption once the W3C has completed its work in this area.

7.6 Using the Person Data Model

7.6.1 Changes from Enterprise Specification v1.1

The changes from the <person> structure in the Enterprise Specification v1.1 are:

- There are no XSD attributes used in the I-PMS. All of the attributes have been replaced by elements (this is to avoid the occurrence of attributes in SOAP messages) and when appropriate the content of the element has been constrained using an enumerated list;
- In the I-PMS binding a 'camel case' naming convention has been adopted. In the Enterprise Specification v1.1 the naming convention was lower case;
- In the I-PMS there are no mandatory children of the <person> element. This is because the 'Create/Read/Update/Delete' approach only needs the data required for the operation e.g., an update of the name does not need any other non-name data;
- The 'recstatsus' attribute in Enterprise Specification v1.1 has been removed. This is now replaced by the service operation definitions;
- The <comments> element, in the Enterprise Specification v1.1, has been replaced by the <recordInfo> element;
- The <sourcedId> element in Enterprise Specification v1.1 has been removed. This information is now passed as a parameter in the request/response messages i.e., it is passed in the SOAP message but not as part of the <person> data model;
- The <sort>, <nickname>, <family>, <given>, <other>, <prefix> and <suffix> elements in the in Enterprise Specification v1.1 have been removed from the <name> element. These have been redundant by the usage of the <partname> element;
- Within the I-PMS some elements are name-spaced entries from a common definitions. These elements are <esx:dataSource>, <esx:email> and <esx:url>;
- In the I-PMS the structure of the <extension> element has been changed. Within the data model, extensions must use the defined layout template. This change was made to ensure that the Service Provider will always be able to unmarshal the received SOAP message;
- Whenever possible data-types in the I-PMS have been used. In some cases in the Enterprise Specification the string data-type was used to contain values that could have been defined as Boolean, etc. Whenever the string data-type has been used an effort has been made to constrain its length in the binding as defined in the information model. The Enterprise Specification v1.1. DTD binding did not constrain the lengths of the strings.

7.6.2 Considerations for Each Operation

Some useful notes to consider when implementing each operation as a Service Provider are:

- CreatePerson – it is possible to create an object that is empty i.e., a 'sourcedId' has been allocated, the base record structure space is reserved but no content is added at the time of creation. The reception of an empty data structure from the Service Requester should not result in the reporting of an error status code;
- CreateByProxyPerson – as per the 'CreatePerson' operation, it is possible to create an object that is empty i.e., a 'sourcedId' is allocated, and the base record structure space reserved but no content is added at the time of creation. The reception of an empty data structure from the Service Requester should not result in the reporting of an error status code;
- DeletePerson – it is implementation dependent as to whether the object is actually destroyed or merely marked as deleted in the server database. It is recommended that no object is destroyed due to the delete request;
- ReadPerson – if the data record is empty then it must still be returned and the success status code reported. The Service provider must return all of the data it stores for the object;
- UpdatePerson – this is an additive operation but the actions on each data structure are determined by the multiplicity defined in the information model i.e., an update for a data structure that has multiplicity of '0' or '1' is equivalent to replacing that structure. The Service Provider should complete as much of the change as possible e.g., if some data is supplied that cannot be stored then this should not result in the complete rejection/failure of the request;

- ReplacePerson – this results in the original data for the identified object being deleted and replaced by this new information. All of the established membership relationships are still maintained because the ‘sourcedId’ for the Person has not changed. The Service Provider should complete as much of the change as possible e.g., if some data is supplied that cannot be stored then this should not result in the complete rejection/failure of the request;
- ChangePersonIdentifier – the successful completion of this request requires that all of the associated membership records must be changed to use the new ‘sourcedId’;
- CreatePersons – look at the notes for the ‘CreatePerson’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- CreateByProxyPersons – look at the notes for the ‘CreateByProxyPerson’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- DeletePersons – look at the notes for the ‘DeletePerson’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReadPersons – look at the notes for the ‘ReadPerson’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReadPersonsForGroup – the Service Provider must return all of the Person records it locates that are members of the defined Group;
- UpdatePersons – look at the notes for the ‘UpdatePerson’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReplacePersons – look at the notes for the ‘ReplacePerson’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ChangePersonsIdentifier – look at the notes for the ‘ChangePersonIdentifier’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction.

Some useful notes to consider when implementing each operation as a Service Requester are:

- CreatePerson – the specification makes no recommendations as to what the Service Requester should do if the create request is rejected by the Service Provider. The nature of the rejection will determine the recovery approach e.g., if the new ‘sourcedId’ has already been allocated in the Service Provider then a change of sourcedId may result in success;
- CreateByProxyPerson – the specification makes no recommendations as to what the Service Requester should do if the ‘sourcedId’ allocated by the Service Provider has already been allocated to another object in the Service Requester. Consistency suggests that the object in the Service Provider should either be deleted, and then perhaps recreated, or have its ‘sourcedId’ changed;
- DeletePerson – it is recommended that the local version of the object not be deleted until confirmation has been received that the Service Provider has successfully completed the deletion request. This avoids the two systems becoming inconsistent;
- ReadPerson – the Service Provider can return an empty data record (see the notes for CreatePerson from the perspective of the Service Provider). The Service Provider may return more information than can be handed by the Service Requester. The Service Requester should supply as much of this data as possible to the invoking application;
- UpdatePerson – the specification makes no recommendations as to what the Service Requester should do if the request is rejected. Some remedial action is required otherwise the system will remain in an inconsistent state;
- ReplacePerson – the specification makes no recommendations as to what the Service Requester should do if the request is rejected. Some remedial action is required otherwise the system will remain in an inconsistent state;

- ChangePersonIdentifier – look at the notes for the ‘CreateByProxyPerson’ operation. The set of status reports in the SOAP message header must be matched to the ‘sourcedId’s returned in the SOAP message body by the Service Provider and the original individual create requests. A status error code for a transaction will mean that there is no corresponding ‘sourcedId’ in the SOAP message body;
- CreatePersons – look at the notes for the ‘CreatePerson’ operation. The set of status reports in the SOAP message header must be matched to the individual create requests;
- CreateByProxyPersons – look at the notes for the ‘CreateByProxyPerson’ operation. The set of status reports in the SOAP message header must be matched to the ‘sourced’s returned in the SOAP message body by the Service Provider and the original individual create requests. A status error code for a transaction will mean that there is no corresponding ‘sourcedId’ in the SOAP message body;
- DeletePersons – look at the notes for the ‘DeletePerson’ operation. The set of status reports in the SOAP message header must be matched to the individual delete requests;
- ReadPersons – look at the notes for the ‘ReadPerson’ operation. The set of status reports in the SOAP message header must be matched to the data returned in the SOAP message body by the Service Provider and the original individual read requests. A status error code for a transaction will mean that there is no corresponding person record in the SOAP message body;
- ReadPersonsForGroup – the Service Provider does not report the number of Person records supplied. Instead it just supplies the records themselves. There is a single status report for this operation;
- UpdatePersons – look at the notes for the ‘UpdatePerson’ operation. The set of status reports in the SOAP message header must be matched to the individual update requests;
- ReplacePersons – look at the notes for the ‘ReplacePerson’ operation. The set of status reports in the SOAP message header must be matched to the individual replace requests;
- ChangePersonsIdentifier – look at the notes for the ‘ChangePersonIdentifier’ operation. The set of status reports in the SOAP message header must be matched to the individual change requests.

7.7 Using the Group Data Model

7.7.1 Changes from Enterprise Specification v1.1

The changes from the <group> structure in the Enterprise Specification v1.1 are:

- There are no XSD attributes used in the I-GMS. All of the attributes have been replaced by elements (this is to avoid the occurrence of attributes in SOAP messages) and when appropriate the content of the element has been constrained using an enumerated list;
- In the I-GMS binding a ‘camel case’ naming convention has been adopted. In the Enterprise Specification v1.1 the naming convention was lower case;
- In the I-GMS there are no mandatory children of the <person> element. This is because the ‘Create/Read/Update/Delete’ approach only needs the data required for the operation e.g., an update of the name does not need any other non-name data;
- The ‘recstatsus’ attribute in Enterprise Specification v1.1 has been removed. This is now replaced by the service operation definitions;
- The <comments> element, in the Enterprise Specification v1.1, has been replaced by the <recordInfo> element;
- The <sourcedId> element in Enterprise Specification v1.1 has been removed as a direct child of the <group> element. This information is now passed as a parameter in the request/response messages i.e., it is passed in the SOAP message but not as part of the <person> data model;
- Within the I-GMS some elements are name-spaced entries from a common definitions. These elements are <esx:dataSource>, <esx:email> and <esx:url>;
- In the I-GMS the structure of the <extension> element has been changed. Within the data model, extensions must use the defined layout template. This change was made to ensure that the Service Provider will always be able to unmarshal the received SOAP message;

- Whenever possible data-types in the I-GMS have been used. In some cases in the Enterprise Specification the string data-type was used to contain values that could have been defined as Boolean, etc. Whenever the string data-type has been used an effort has been made to constrain its length in the binding as defined in the information model. The Enterprise Specification v1.1. DTD binding did not constrain the lengths of the strings.

7.7.2 Considerations for Each Operation

Some useful notes to consider when implementing each operation as a Service Provider are:

- CreateGroup – it is possible to create an object that is empty i.e., a ‘sourcedId’ has been allocated, the base record structure space is reserved but no content is added at the time of creation. The reception of an empty data structure from the Service Requester should not result in the reporting of an error status code;
- CreateByProxyGroup – as per the ‘CreateGroup’ operation, it is possible to create an object that is empty i.e., a ‘sourcedId’ is allocated, and the base record structure space reserved but no content is added at the time of creation. The reception of an empty data structure from the Service Requester should not result in the reporting of an error status code;
- DeleteGroup – it is implementation dependent as to whether the object is actually destroyed or merely marked as deleted in the server database. It is recommended that no object be destroyed due to the delete request. The deletion of a group requires that all of the child sub-groups are also deleted and the associated membership records of the Group are deleted;
- ReadGroup – if the data record is empty then it must still be returned and the success status code reported. The Service provider must return all of the data it stores for the object. Only the record of the identified group is returned i.e., information about related groups requires further read requests using the appropriate ‘sourcedId’;
- UpdateGroup – this is an additive operation but the actions on each data structure are determined by the multiplicity defined in the information model i.e., an update for a data structure that has multiplicity of ‘0’ or ‘1’ is equivalent to replacing that structure. The Service Provider should complete as much of the change as possible e.g., if some data is supplied that cannot be stored then this should not result in the complete rejection/failure of the request;
- ReplaceGroup – this results in the original data for the identified object being deleted and replaced by this new information. All of the established membership relationships are still maintained because the ‘sourcedId’ for the Group has not changed. All of the child sub-groups are deleted. The Service Provider should complete as much of the change as possible e.g., if some data is supplied that cannot be stored then this should not result in the complete rejection/failure of the request;
- ChangeGroupIdentifier – the successful completion of this request requires that all of the associated membership records must be changed to use the new ‘sourcedId’;
- CreateGroups – look at the notes for the ‘CreateGroup’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- CreateByProxyGroups – look at the notes for the ‘CreateByProxyGroup’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- DeleteGroups – look at the notes for the ‘DeleteGroup’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReadGroups – look at the notes for the ‘ReadGroup’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReadGroupsForPerson – the Service Provider must return all of the Group records it locates that are members of the defined Person;
- UpdateGroups – look at the notes for the ‘UpdateGroup’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;

- ReplaceGroups – look at the notes for the ‘ReplaceGroup’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ChangeGroupsIdentifier – look at the notes for the ‘ChangeGroupIdentifier’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction.

Some useful notes to consider when implementing each operation as a Service Requester are:

- CreateGroup – the specification makes no recommendations as to what the Service Requester should do if the create request is rejected by the Service Provider. The nature of the rejection will determine the recovery approach e.g., if the new ‘sourcedId’ has already been allocated in the Service Provider then a change of sourcedId may result in success;
- CreateByProxyGroup – the specification makes no recommendations as to what the Service Requester should do if the ‘sourcedId’ allocated by the Service Provider has already been allocated to another object in the Service Requester. Consistency suggests that the object in the Service Provider should either be deleted, and then perhaps recreated, or have its ‘sourcedId’ changed;
- DeleteGroup – it is recommended that the local version of the object not be deleted until confirmation has been received that the Service Provider has successfully completed the deletion request. This avoids the two systems becoming inconsistent;
- ReadGroup – the Service Provider can return an empty data record (see the notes for CreateGroup from the perspective of the Service Provider). The Service Provider may return more information than can be handed by the Service Requester. The Service Requester should supply as much of this data as possible to the invoking application;
- UpdateGroup – the specification makes no recommendations as to what the Service Requester should do if the request is rejected. Some remedial action is required otherwise the system will remain in an inconsistent state;
- ReplaceGroup – the specification makes no recommendations as to what the Service Requester should do if the request is rejected. Some remedial action is required otherwise the system will remain in an inconsistent state;
- ChangeGroupIdentifier – the specification makes no recommendations as to what the Service Requester should do if the request is rejected because the new ‘sourcedId’ has already been allocated to another object in the Service Provider or if the original object cannot be identified in the Service Provider;
- CreateGroups – look at the notes for the ‘CreateGroup’ operation. The set of status reports in the SOAP message header must be matched to the individual create requests;
- CreateByProxyGroups – look at the notes for the ‘CreateByProxyGroup’ operation. The set of status reports in the SOAP message header must be matched to the ‘sourcedId’s returned in the SOAP message body by the Service Provider and the original individual create requests. A status error code for a transaction will mean that there is no corresponding ‘sourcedId’ in the SOAP message body;
- DeleteGroups – look at the notes for the ‘DeleteGroup’ operation. The set of status reports in the SOAP message header must be matched to the individual delete requests;
- ReadGroups – look at the notes for the ‘ReadGroup’ operation. The set of status reports in the SOAP message header must be matched to the data returned in the SOAP message body by the Service Provider and the original individual read requests. A status error code for a transaction will mean that there is no corresponding person record in the SOAP message body;
- ReadGroupsForPerson – the Service Provider does not report the number of Group records supplied. Instead it just supplies the records themselves. There is a single status report for this operation;
- UpdateGroups – look at the notes for the ‘UpdateGroup’ operation. The set of status reports in the SOAP message header must be matched to the individual update requests;
- ReplaceGroups – look at the notes for the ‘ReplaceGroup’ operation. The set of status reports in the SOAP message header must be matched to the individual replace requests;
- ChangeGroupsIdentifier – look at the notes for the ‘ChangeGroupIdentifier’ operation. The set of status reports in the SOAP message header must be matched to the individual change requests.

7.7.2 Groups and Sub-groups

The Group structure is used as a generic collection structure for objects that have a similar functionality, responsibility, etc. It can be a Group of other Groups or a Group of Person records. Sub-groups are defined by creating a relationship between Group and another and by establishing the relationship as either 'Parent' or 'Child'.

The relationship between two Groups is defined using the <relationship> element. This allows the definition of uni-directional and bi-directional references depending on the desired implementation. The sub-group hierarchy is defined using the '1=Parent' and '2=Child' values in the <relation> attribute of the <relationship> element. This tree structure can be defined to have any number of roots and the equivalence of Groups can be defined using the '3-Cross Listing' value for <relation>.

7.7.3 Cross-listed Course Sections

Cross-Listed course sections are a fairly common scenario in higher education. A Cross-Listed course section refers to a situation where the same course is offered under more than one name. This is typically done because different groups of students will enroll in different sections based on the program they are studying. For example, Statistics 101 section 1 and Psychology 101 section 1 are really the same course section, offered by the same instructor, meeting at the same time and place (physical or virtual), using the same course materials. The only difference is that Math students enroll in Statistics 101, and Psychology students enroll in Psychology 101. A problem arises in this situation when the Enterprise system treats these sections as separate groups. In the Learning Management System, they need to be treated as a single group, or at least the LMS needs to know they are related.

One approach is to resolve this in the source system before passing Groups and Memberships over to the target system. In other words, a single group (perhaps called "Introductory Statistics", without the Math or Psychology designator) would be created in the Learning Management System and membership from both groups in the Enterprise system would be passed to this single group in the LMS. Another approach is to pass two separate groups to the target system, but relate them to each other through the use of the Relationship element. In this case, the two groups could be tagged as follows in the Relationship element:

- Sourced ID – The ID of the cross-listed course section in the source system;
- Label – Would contain something like 'Cross Listed Section';
- Relation – Would contain "3" (also known as).

In general, the best practice would be to resolve the issue in the source system, before passing the group and membership data to the target system, but there may be cases where the second approach is required.

7.8 Using the Membership Data Model

The changes from the <membership> structure in the Enterprise Specification v1.1 are:

- There are no XSD attributes used in the I-MMS. All of the attributes have been replaced by elements (this is to avoid the occurrence of attributes in SOAP messages) and when appropriate the content of the element has been constrained using an enumerated list;
- In the I-MMS binding a 'camel case' naming convention has been adopted. In the Enterprise Specification v1.1 the naming convention was lower case;
- In the I-MMS there are no mandatory children of the <person> element. This is because the 'Create/Read/Update/Delete' approach only needs the data required for the operation e.g., an update of the name does not need any other non-name data;
- In the I-MMS the <membership> element can only contain one <member> record but this may contain more than one <role> record;
- In the I-MMS each Membership record is allocated its own 'sourcedId'. All operations on the Membership record must use this 'sourcedId'. In the Enterprise Specification v1.1 a Membership record was referenced using the 'sourcedid's of the appropriate Member and Group;
- The 'recstatsus' attribute in Enterprise Specification v1.1 has been removed. This is now replaced by the service operation definitions;

- The <comments> element, in the Enterprise Specification v1.1, has been replaced by the <recordInfo> element;
- The <sourcedId> element in Enterprise Specification v1.1 has been removed. This information is now passed as a parameter in the request/response messages i.e., it is passed in the SOAP message but not as part of the <person> data model;
- Within the I-MMS some elements are name-spaced entries from a common definitions. These elements are <esx:dataSource> and <esx:email>;
- In the I-MMS the structure of the <extension> element has been changed. Within the data model, extensions must use the defined layout template. This change was made to ensure that the Service Provider will always be able to unmarshal the received SOAP message;
- Whenever possible data-types in the I-MMS have been used. In some cases in the Enterprise Specification the string data-type was used to contain values that could have been defined as Boolean, etc. Whenever the string data-type has been used an effort has been made to constrain its length in the binding as defined in the information model. The Enterprise Specification v1.1. DTD binding did not constrain the lengths of the strings.

7.8.1 Considerations for Each Operation

Some useful notes to consider when implementing each operation as a Service Provider are:

- CreateMembership – it is possible to create an object that is empty i.e., a ‘sourcedId’ has been allocated, the base record structure space is reserved but no content is added at the time of creation. The reception of an empty data structure from the Service Requester should not result in the reporting of an error status code;
- CreateByProxyMembership – as per the ‘CreatePerson’ operation, it is possible to create an object that is empty i.e., a ‘sourcedId’ is allocated, and the base record structure space reserved but no content is added at the time of creation. The reception of an empty data structure from the Service Requester should not result in the reporting of an error status code;
- DeleteMembership – it is implementation dependent as to whether the object is actually destroyed or merely marked as deleted in the server database. It is recommended that no object is destroyed due to the delete request. Only the membership record is deleted;
- ReadMembership – if the data record is empty then it must still be returned and the success status code reported. The Service provider must return all of the data it stores for the object. Only the Membership record is returned;
- UpdateMembership – this is an additive operation but the actions on each data structure are determined by the multiplicity defined in the information model i.e., an update for a data structure that has multiplicity of ‘0’ or ‘1’ is equivalent to replacing that structure. The Service Provider should complete as much of the change as possible e.g., if some data is supplied that cannot be stored then this should not result in the complete rejection/failure of the request;
- ReplaceMembership – this results in the original data for the identified object being deleted and replaced by this new information. All of the established membership relationships are still maintained because the ‘sourcedId’ for the Person has not changed. The Service Provider should complete as much of the change as possible e.g., if some data is supplied that cannot be stored then this should not result in the complete rejection/failure of the request;
- ChangeMembershipIdentifier – the successful completion of this request requires that all of the associated membership records must be changed to use the new ‘sourcedId’;
- CreateMemberships – look at the notes for the ‘CreateMembership’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- CreateByProxyMemberships – look at the notes for the ‘CreateByProxyMembership’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- DeleteMemberships – look at the notes for the ‘DeleteMembership’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReadMemberships – look at the notes for the ‘ReadMembership’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;

- ReadMembershipsForGroup – the Service Provider must return all of the Membership records it locates that are members of the defined Group;
- ReadMembershipsForPerson – the Service Provider must return all of the Membership records it locates that are members of the defined Person;
- UpdateMemberships – look at the notes for the ‘UpdateMembership’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ReplaceMemberships – look at the notes for the ‘ReplaceMembership’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction;
- ChangeMembershipsIdentifier – look at the notes for the ‘ChangeMembershipIdentifier’ operation. A status report must be supplied for every transaction in the request otherwise the Service Requester may not be able to accurately determine the status of any transaction.

Some useful notes to consider when implementing each operation as a Service Requester are:

- CreateMembership – the specification makes no recommendations as to what the Service Requester should do if the create request is rejected by the Service Provider. The nature of the rejection will determine the recovery approach e.g., if the new ‘sourcedId’ has already been allocated in the Service Provider then a change of sourcedId may result in success;
- CreateByProxyMembership – the specification makes no recommendations as to what the Service Requester should do if the ‘sourcedId’ allocated by the Service Provider has already been allocated to another object in the Service Requester. Consistency suggests that the object in the Service Provider should either be deleted, and then perhaps recreated, or have its ‘sourcedId’ changed;
- DeleteMembership – it is recommended that the local version of the object not be deleted until confirmation has been received that the Service Provider has successfully completed the deletion request. This avoids the two systems becoming inconsistent;
- ReadMembership – the Service Provider can return an empty data record (see the notes for CreateMembership from the perspective of the Service Provider). The Service Provider may return more information than can be handed by the Service Requester. The Service Requester should supply as much of this data as possible to the invoking application;
- UpdateMembership – the specification makes no recommendations as to what the Service Requester should do if the request is rejected. Some remedial action is required otherwise the system will remain in an inconsistent state;
- ReplaceMembership – the specification makes no recommendations as to what the Service Requester should do if the request is rejected. Some remedial action is required otherwise the system will remain in an inconsistent state;
- ChangeMembershipIdentifier – the specification makes no recommendations as to what the Service Requester should do if the request is rejected because the new ‘sourcedId’ has already been allocated to another object in the Service Provider or if the original object cannot be identified in the Service Provider;
- CreateMemberships – look at the notes for the ‘CreateMembership’ operation. The set of status reports in the SOAP message header must be matched to the individual create requests;
- CreateByProxyMemberships – look at the notes for the ‘CreateByProxyMembership’ operation. The set of status reports in the SOAP message header must be matched to the ‘sourcedId’s returned in the SOAP message body by the Service Provider and the original individual create requests. A status error code for a transaction will mean that there is no corresponding ‘sourcedId’ in the SOAP message body;
- DeleteMemberships – look at the notes for the ‘DeleteMembership’ operation. The set of status reports in the SOAP message header must be matched to the individual delete requests;
- ReadMemberships – look at the notes for the ‘ReadMembership’ operation. The set of status reports in the SOAP message header must be matched to the data returned in the SOAP message body by the Service Provider and the original individual read requests. A status error code for a transaction will mean that there is no corresponding person record in the SOAP message body;
- ReadMembershipsForGroup – the Service Provider does not report the number of Membership records supplied. Instead it just supplies the records themselves. There is a single status report for this operation;

- ReadMembershipsForPerson – the Service Provider does not report the number of Membership records supplied. Instead it just supplies the records themselves. There is a single status report for this operation;
- UpdateMemberships – look at the notes for the ‘UpdateMembership’ operation. The set of status reports in the SOAP message header must be matched to the individual update requests;
- ReplaceMemberships – look at the notes for the ‘ReplaceMembership’ operation. The set of status reports in the SOAP message header must be matched to the individual replace requests;
- ChangeMembershipsIdentifier – look at the notes for the ‘ChangeMembershipIdentifier’ operation. The set of status reports in the SOAP message header must be matched to the individual change requests.

7.8.2 Assigning Group Membership Role-type

Roletype is a data structure in the Group Membership object that has a defined set of domain values. This means that only those values defined in the domain can be used for this element. Recognizing that no defined list of roles can ever be absolutely complete; the optional element <subrole> can be used to further qualify a person’s role in a group. It is essential to have a defined list of values for the mandatory <roletype> element so source systems can generate standard Group Membership data objects that target systems can process without having to first negotiate the meaning of role-types with the source system. To help developers understand what meaning is embedded in each of the role-type values, the Table 7.1 shows the LMS functions to which each <roletype> will ***typically*** have access. This is not intended to be a precise and exclusive list of all functions that these roles will have access to in all LMSs. Rather, it is provided as an interpretive guide intended to communicate the meaning the developers of the specification had in mind for each role. In addition, access to these functions will be less for some sub-roles. For example, a supervisor may be a sub-role for a manager, and a supervisor will likely not have access to results for the people they supervise.

Table 7.1 The LMS functions typically available to each roletype.

	01 Learner		**02 Instructor**		**03 Content Developer**		**04 Member**		**05 Manager**		**06 Mentor**		**07 Administrator**		**08 Teaching Assistant**	
	R	**M**	**R**	**M**	**R**	**M**	**R**	**M**	**R**	**M**	**R**	**M**	**R**	**M**	**R**	**M**
Learning Content	X		X		X	X	X		X		X		X	X	X	X
Learner Enrollment	X	T	X	X			X	T					X	X	X	
Group Roles	X		X				X		X		X		X	X	X	
Learner Submission	T	T	X						S		S		X			
Unofficial Results	T		X	X					S		S		X	X	X	X
Official Results	T		X	X					S		S		X	X	R	
Final Results	T		X	X					S		S		X	X	R	
Certification	T		X	X					S		S		X	X	X	

The key is: X=Available, R=Read, M=Main, S=Some, T=Theirs.

In any particular implementation or in any specific vendor’s product, the Instructor role and the Administrator role will frequently have several sub-roles.

In later versions of the IMS Enterprise Service we will be revisiting the ways in which the role-type is defined and referenced. The changes will look at using a vocabulary that will be supported using the IMS VDEX specification [VDEX, 04]. The current method is maintained for compatibility as this was used in the IMS Enterprise Specification v1.1.

7.9 IMS Harmonization

The IMS Enterprise specification is complemented by the IMS LIP specification [LIP, 01a], [LIP, 01b], [LIP, 01c]. There is some confusion about when and where the two specifications should be used and so the following recommendations are made:

- Information that is limited to a small sub-set of the full personal details of an individual and to be used to populate an LMS for learning activities should be exchanged using the IMS Person Management Service. The exception is personal information required to support computer-based accessibility that should be exchanged using the IMS LIP <accessibility> data object;
- Information about Group activities, groups of people and membership information should be exchanged using the IMS Group Management Services and IMS Membership Management Services specifications;
- Information about a single individual, particularly when focused on their life-long learning log/profile, should be exchanged using the IMS LIP specification e.g., their set of examination results.

There may be some confusion between the IMS Enterprise, IMS LIP and the IMS Question & Test Interoperability (QTI) Results Reporting specifications with respect to exchanging assessment results. The following recommendations are made:

- Information from an Assessment Engine reporting the set of results for a single individual or group of people should be reported to another Assessment Engine or within the Assessment System using the IMS QTI specification;
- Personal results being reported from an Assessment System to a Profiles System should be exchanged using the IMS LIP;
- Personal and group results being reported from and LMS to an Enterprise System should be reported using the IMS Membership Management Services.

8. Supporting the Use Cases

8.1 Person Management Service Use Cases

The manner in which the original ES Use Cases of the Person Management Service [EntServices, 04a] are supported is shown in Table 8.1.

Table 8.1 How the original Person use cases are supported by the specification.

Original PMS Use Case	Manner of Support
Person01/01–Creating a Person	If the 'Reference Agent' is responsible for allocating the unique identifier (SourcedId) then it issues the 'createPerson' request. The 'Sync Agent' replies with a 'createPerson' response. If the 'Sync Agent' is responsible for allocating the unique identifier (SourcedId) then the 'Reference Agent' issues the 'createByProxyPerson' request. The 'Sync Agent' replies with a 'createByProxyPerson' response.
Person01/02–Reading a Person	The 'Reference Agent' issues the 'readPerson' request to the 'Sync Agent'. The 'Sync Agent' replies using the 'readPerson' response.
Person01/03–Updating a Person	If the 'Reference Agent' requires the addition of new information to the equivalent record on the 'Sync Agent' then an 'updatePerson' request is issued. The 'Sync Agent' replies with the 'updatePerson' response. However, if the 'Reference Agent' requires the equivalent record on the 'Sync Agent' to be replaced by a new set of data then the 'replacePerson' request is issued. The 'Sync Agent' replies with the 'replacePerson' response'.
Person01/04–Deleting a Person	The 'Reference Agent' issues the 'deletePerson' request to the 'Sync Agent'. The 'Sync Agent' replies using the 'deletePerson' response.
Person01/05–Changing a Person Source Identifier	The 'Reference Agent' issues the 'changePersonIdentifier' request to the 'Sync Agent'. The 'Sync Agent' replies with the 'changePersonIdentifier' response. Note that it is not possible to support the part of the use case that requires the change of identifiers since a specific time.
Person02/01–Querying a Person using Creation Date	This use case is not supported in the Version 1.0 release of this specification.
Person02/02–Querying Updated Persons	This use case is not supported in the Version 1.0 release of this specification.
Person02/03–Querying Deleted Persons	This use case is not supported in the Version 1.0 release of this specification.

8.2 Group Management Service Use Cases

The manner in which the original ES Use Cases of the Group Management Service [EntServices, 04a] are supported is shown in Table 8.2.

Table 8.2 How the original Group use cases are supported by the specification.

Original GMS Use Case	Manner of Support
Group01/01–Creating a Group	If the ‘Reference Agent’ is responsible for allocating the unique identifier (SourcedId) then it issues the ‘createGroup’ request. The ‘Sync Agent’ replies with a ‘createGroup’ response. If the ‘Sync Agent’ is responsible for allocating the unique identifier (SourcedId) then the ‘Reference Agent’ issues the ‘createByProxyGroup’ request. The ‘Sync Agent’ replies with a ‘createByProxyGroup’ response.
Group01/02–Reading a Group	The ‘Reference Agent’ issues the ‘readGroup’ request to the ‘Sync Agent’. The ‘Sync Agent’ replies using the ‘readGroup’ response.
Group01/03–Updating a Group	If the ‘Reference Agent’ requires the addition of new information to the equivalent record on the ‘Sync Agent’ then an ‘updateGroup’ request is issued. The ‘Sync Agent’ replies with the ‘updateGroup’ response. However, if the ‘Reference Agent’ requires the equivalent record on the ‘Sync Agent’ to be replaced by a new set of data then the ‘replaceGroup’ request is issued. The ‘Sync Agent’ replies with the ‘replaceGroup’ response’.
Group01/04–Deleting a Group	The ‘Reference Agent’ issues the ‘deleteGroup’ request to the ‘Sync Agent’. The ‘Sync Agent’ replies using the ‘deleteGroup’ response.
Group01/05–Deleting a Group Relationship	The ‘Reference Agent’ issues the ‘deleteGroupRelationship’ request to the ‘Sync Agent’. The ‘Sync Agent’ replies using the ‘deleteGroupRelationship’ response. Note that it is not possible to support the part of the use case that requires the deletion of all relationships since a specific time.

8.3 Membership Management Service Use Cases

The manner in which the original ES Use Cases of the Membership Management Service [EntServices, 04a] are supported is shown in Table 8.3.

Table 8.3 How the original Membership use cases are supported by the specification.

Original MMS Use Case	Manner of Support
Membership01/01–Creating a Membership	If the ‘Reference Agent’ is responsible for allocating the unique identifier (SourcedId) then it issues the ‘createMembership’ request. The ‘Sync Agent’ replies with a ‘createMembership’ response. If the ‘Sync Agent’ is responsible for allocating the unique identifier (SourcedId) then the ‘Reference Agent’ issues the ‘createByProxyMembership’ request. The ‘Sync Agent’ replies with a ‘createByProxyMembership’ response.
Membership01/02–Reading a Membership	The ‘Sync Agent’ issues the ‘readMembership’ request to the ‘Reference Agent’. The ‘Reference Agent’ replies using the ‘readMembership’ response.
Membership01/03–Updating a Membership	If the ‘Reference Agent’ requires the addition of new information to the equivalent record on the ‘Sync Agent’ then an ‘updateMembership’ request is issued. The ‘Sync Agent’ replies with the ‘updateMembership’ response. However, if the ‘Reference Agent’ requires the equivalent record on the ‘Sync Agent’ to be replaced by a new set of data then the ‘replaceMembership’ request is issued. The ‘Sync Agent’ replies with the ‘replaceMembership’ response’.
Membership01/04–Deleting a Membership	The ‘Reference Agent’ issues the ‘deleteMembership’ request to the ‘Sync Agent’. The ‘Sync Agent’ replies using the ‘deleteMembership’ response.

Original MMS Use Case	**Manner of Support**
Membership01/05–Reading a Person’s Membership	The ‘Sync Agent’ issues the ‘readPersonsForMembership’ request and the ‘Reference Agent’ replies with the ‘readPersonsForMembership’ response.
Membership01/06–Reading a Group’s Membership	The ‘Sync Agent’ issues the ‘readGroupsForMembership’ request and the ‘Reference Agent’ replies with the ‘readGroupsForMembership’ response.

9. Extending the Services

9.1 Proprietary Extensions to the Enterprise Services

9.1.1 Extensions to the Data Models

Extensions in a data model must use the IMS Extension class. As an example consider an extension to the Person model in which the color of the person's eyes and hair is to be stored:

```
0001   POST /PersonManagementService HTTP/1.1
0002   Host: www.personmanagementserver.com
0003   Content-Type: text/xml; charset="utf-8"
0004   Content-Length: nnnn
0005   SOAPAction: "http://www.imsglobal.org/soap/pms/createPerson"
0006
0007   <SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
0008      <SOAP-ENV:Header>
0009         <h:syncRequestHeaderInfo xmlns:h="../imsMessBindSchemav1p0.xsd">
0010            <h:messageIdentifier>AB12345e4t6789</h:messageIdentifier>
0011         </h:syncRequestHeaderInfo>
0012      </SOAP-ENV:Header>
0013      <SOAP-ENV:Body>
0014         <m:createPersonRequest xmlns:m="../imsPersonManMessSchemav1p0.xsd">
0015            <m:sourcedId>
0016               <esx:identifier>oldsource:oldidentifier</esx:identifier>
0017            </m:sourcedId>
0018            <m:person>
0019               <esx:email xmlns:esx = "http://…">hello</esx:email>
0020               <per:extension xmlns:per = "http://…">
0021                  <esx:extensionField xmlns:esx = "http://…">
0022                     <esx:fieldName>eyeColour</esx:fieldName>
0023                     <esx:fieldType>String</esx:fieldType>
0024                     <esx:fieldValue>Blue</esx:fieldValue>
0025                  </esx:extensionField>
0026                  <esx:extensionField xmlns:esx = "http://…">
0027                     <esx:fieldName>hairColour</esx:fieldName>
0028                     <esx:fieldType>String</esx:fieldType>
0029                     <esx:fieldValue>Black</esx:fieldValue>
0030                  </esx:extensionField>
0031               </per:extension>
0032            </m:person>
0033         </m:createPersonRequest>
0034      </SOAP-ENV:Body>
0035   </SOAP-ENV:Envelope>
```

The extensions are shown in lines 20-31. The points to note are:

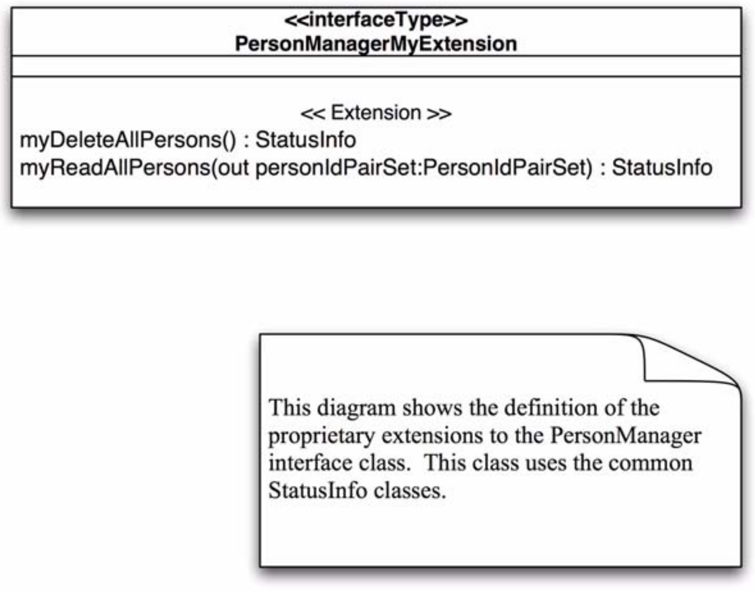
- There is no requirement to define any new XSD structures;
- There is no hierarchical structure of the extensions. The extension structure must be expressed as a flat list;
- Any conditional relationship between the sub-elements are not validated by the parser e.g., if the <fieldType> was an enumerated list then the parser will not ensure that the <fieldValue> is an entry from that list.

9.1.2 Extensions to the Behaviors

The addition of proprietary behaviors should still maintain the format of the established behaviors. An example is shown in Figure 9.1 in which two new behaviors are defined (these use the current set of data classes). When new proprietary behaviors are added it is recommended that:

- A new interface type is declared and this has a unique name that indicates it is an extension;

- Each new behavior should have a unique name. All of the behaviors must return a status object either in the form of 'StatusInfo' or 'StatusInfoSet';
- Any new data classes should also be described. These new classes should be given their own namespace and linked, where appropriate, to the current XSDs;
- The corresponding new WSDL files must be created (these will extend but replace the original WSDL files). Each new behavior requires the appropriate message choreography (depending on the nature of the binding) and must use the established SOAP message header structures. The established transformation rules should be used [GWS, 04a], [GWS, 04b].



<<interfaceType>>
PersonManagerMyExtension
<< Extension >>
myDeleteAllPersons() : StatusInfo
myReadAllPersons(out personIdPairSet:PersonIdPairSet) : StatusInfo
This diagram shows the definition of the proprietary extensions to the PersonManager interface class. This class uses the common StatusInfo classes.

Figure 9.1 Extending a service by adding proprietary behaviors.

Note that a proprietary extension MUST NOT be defined to replace the behavior of an established operation. Instead it is permissible to add a new behavior that inherits its functionality form an established behavior.

9.2 Planned Extensions for the Enterprise Services

9.2.1 Potential New Behaviors for Established Services

There are three types of new behaviors that can be added to the current services, namely:

- Query behaviors – this is the capability to query the 'Service Provider' for objects that fit a search criteria e.g., list all Person objects that were changed since yesterday, etc. The query capability is to be added to all three of the current component Enterprise Services. IMS is to develop an independent Query Service that will then be profiled for usage in the Person, Group and Membership Management services. The Query Service will not be available until May 2005 and so any new behaviors in the Enterprise Service will not be supplied until October 2005 at the earliest;
- Object management behaviors – this is the inclusion of more behaviors that support the low level management of the core objects. This could include the ability to delete all objects, read all objects, undelete an object, etc.;
- Business specific behaviors – new behaviors could be created to handle specific types of Person or Group e.g., create new student, create new course, etc. These behaviors would be based upon refinement and combination of some of the other behaviors for a service.

9.2.2 Potential New Services

In the original evaluation of the IMS Enterprise v1.1 specification two new services were investigated for potential inclusion in V1.0 of the IMS Enterprise Services specification. The two new services considered were:

- Grade-book Management Service – this is a considerable extension of the simple results reporting capability currently available in the membership Management Service. A separate service would enable the creation of a Grade-book Management capability that could be designed to be integrated with the IMS Learner Information Package and the SIF Grade-book object;
- Course Catalog Management service – this is a development of a service capability to manage course catalogs, which themselves could consist of sub-catalogs etc. It is possible to provide a simplified capability using the Group Management Service but as yet there is no agreed approach for interoperability.

Other new business services could be added that reflect specific business operations. These new services could be based upon the combination of behaviors from the other services.

Appendix A – Glossary of Terms

Throughout the Enterprise Services specification documents a variety of key terms, concepts and descriptions have been introduced. These terms, concepts and descriptions and defined below but where appropriate the normative definition from the IAF Glossary is referenced [AbsGloss, 03].

address	The *address* element is a part of the *person* element. The address contains information such as the street, region, country, zip code, etc.
AddressDType	The *AddessDType* is an XSD complexType that is defined to act as the data type for address structures. The *address* element is of type *AddressDType*.
adminPeriod	The *adminPeriod* element is used within the *timeFrame* element to contain a short human readable description for the administrative or academic period for the duration of the Group. It is a character string [1-32].
authenticationType	The *authenticationType* element is used on the *userId* element to identify the authentication mechanism to be applied to control the user’s access to the learning environment. It is a character string [1-32].
bday	The *bday* element is used within the *demographics* element to store the date of birth of the Person. This is a character string containing the date in the ISO8601 format (YYYY-MM-DD).
begin	The *begin* element is used within the *timeFrame* element to define when a Group is to be made available for participation. It is a character string [1-20] containing the date in the ISO8601 format (YYYY-MM-DD).
changeGroupIdentifier	This is the behavior to change the *SourcedId* assigned to a *Group* object in the *Group Management Service*. If the new *SourcedId* is already in use or the original *Group* object cannot be identified by the *Service Provider* then a failure status code is returned.
changeGroupsIdentifier	This is the behavior to change the *SourcedIds* assigned to a set of *Group* objects in the *Group Management Service*. If the new *SourcedId* is already in use or the original *Group* object cannot be identified by the *Service Provider* then a failure status code is returned for that part of the request.
changeMembershipIdentifier	This is the behavior to change the *SourcedId* assigned to a *Membership* object in the *Membership Management Service*. If the new *SourcedId* is already in use or the original *Membership* object cannot be identified by the *Service Provider* then a failure status code is returned.
changeMembershipsIdentifier	This is the behavior to change the *SourcedIds* assigned for a set *Membership* objects in the *Membership Management Service*. If the new *SourcedId* is already in use or the original *Membership* object cannot be identified by the *Service Provider* then a failure status code is returned for that part of the request.
changePersonIdentifier	This is the behavior to change the *SourcedId* assigned to a *Person* object in the *Person Management Service*. If the new *SourcedId* is already in use or the original *Person* object cannot be identified by the *Service Provider* then a failure status code is returned.

changePersonsIdentifier	This is the behavior to change the *SourcedIds* assigned to a set of *Person* objects in the *Person Management Service*. If the new *SourcedId* is already in use or the original *Person* object cannot be identified by the *Service Provider* then a failure status code is returned for that part of the request.
codeMajor	This is an attribute of the *StatusInfo* class and a child element of *statusInfo*. It is used to contain the primary status indicator and as such is an enumerated set of 'Success', 'Processing',v'Failure' and 'Unsupported' codes. This is defined as part of the Common Data Definitions.
codeMinor	This is the element that contains the set of minor codes that are returned as part of the *statusInfo*. This element contains one or more *codeMinorField* elements. This is defined as part of the Common Data Definitions.
codeMinorField	This class and element is use to contain the individual minor status codes that are returned in the *statusInfo* and *statusInfoSet* element and classes. This element contains the name and value of the code in the *codeMinorName* and *codeMinorValue* elements respectively. This is defined as part of the Common Data Definitions.
codeMinorName	This element within the *codeMinorField* contains the name of the service that is responsible for generating the status code. This is a character string [1-32]. This is defined as part of the Common Data Definitions.
codeMinorValue	This element within the *codeMinorField* contains the minor status code value being reported. This is a character string [1-32]. This is defined as part of the Common Data Definitions.
comments	The *comments* element is used as the general element in which comments and/or statements are supplied. When used as a comment this information is parsed through the XML parser (unlike and XML comment). This is a character string [1-2048].
country	The *country* element is used within the *address* element. It is used to store the country component of the address e.g., Brazil, China, etc. It is submitted as a string of up to 64 characters in length and the content is based upon the ISO3166 standard.
createByProxyGroup	This is the create *Group* object behavior in the *Group Management Service*. The successful outcome is the creation of a single populated *Group* object on the *Service Provider.* The unique *SourcedId* is allocated by the *Service Provider* and returned to the *Service Requester.*
createByProxyGroups	This is the behavior to create many *Group* objects in the *Group Management Service*. The successful outcome is the creation of a set of populated *Group* objects on the *Service Provider.* The unique *SourcedId*s are allocated by the *Service Provider* and returned to the *Service Requester.* An individual status code is returned for each transaction requested within the operation.
createByProxyMembership	This is the create *Membership* object behavior in the *Membership Management Service*. The successful outcome is the creation of a single populated *Membership* object on the *Service Provider.* The unique *SourcedId* is allocated by the *Service Provider* and returned to the *Service Requester.* This behavior requires valid *Group* and *Member* SourcedIds to be supplied.

createByProxyMemberships	This is the behavior to create many *Membership* objects in the *Membership Management Service*. The successful outcome is the creation of a set of populated *Membership* objects on the *Service Provider.* The unique *SourcedId*s are allocated by the *Service Provider* and returned to the *Service Requester.* An individual status code is returned for each transaction requested within the operation. This behavior also requires valid *Group* and *Member SourcedIds* to be supplied for each *Membership* object being created.
createByProxyPerson	This is the create *Person* object behavior in the *Person Management Service*. The successful outcome is the creation of a single populated *Person* object on the *Service Provider.* The unique *SourcedId* is allocated by the *Service Provider* and returned to the *Service Requester.*
createByProxyPersons	This is the behavior to create many *Person* objects in the *Person Management Service*. The successful outcome is the creation of a set of populated *Person* objects on the *Service Provider.* The unique *SourcedId*s are allocated by the *Service Provider* and returned to the *Service Requester.* An individual status code is returned for each transaction requested within the operation.
createGroup	This is the create *Group* object behavior in the *Group Management Service*. The successful outcome is the creation of a single populated *Group* object on the *Service Provider.* The unique *SourcedId* is allocated by the *Service Requester.* If the allocated *SourcedId* is already in use in the *Service Provider* then a failure status is returned.
createGroups	This is the create many *Group* objects behavior in the *Group Management Service*. The successful outcome is the creation of a set of populated *Group* objects on the *Service Provider.* The unique *SourcedId*s are allocated by the *Service Requester.* If an allocated *SourcedId* is already in use in the *Service Provider* then a failure status is returned for that part of the request.
createMembership	This is the create *Membership* object behavior in the *Membership Management Service*. The successful outcome is the creation of a single populated *Membership* object on the *Service Provider.* The unique *SourcedId* is allocated by the *Service Requester.* If the allocated *SourcedId* is already in use in the *Service Provider* then a failure status is returned. This behavior also requires valid *Group* and *Member* SourcedIds to be supplied.
createMemberships	This is the behavior to create many *Membership* objects in the *Membership Management Service*. The successful outcome is the creation of a set of populated *Membership* objects on the *Service Provider.* The unique *SourcedId*s are allocated by the *Service Requester.* If an allocated *SourcedId* is already in use in the *Provider* then a failure status is returned for that part of the request. This behavior also requires valid *Group* and *Member* SourcedIds to be supplied for each *Membership* object being created.
createPerson	This is the create *Person* object behavior in the *Person Management Service*. The successful outcome is the creation of a single populated *Person* object on the *Service Provider.* The unique *SourcedId* is allocated by the *Service Requester.* If the allocated *SourcedId* is already in use in the *Service Provider* then a failure status is returned.

createPersons	This is the behavior to create many *Person* objects in the *Person Management Service*. The successful outcome is the creation of a set of populated *Person* objects on the *Service Provider.* The unique *SourcedId*s are allocated by the *Service Requester.* If an allocated *SourcedId* is already in use in the *Service Provider* then a failure status is returned for that part of the request.
dataSource	The *dataSource* element is used in the *person*, *group* and *role* elements to indicate the original source of the data i.e., this could be different from the system currently attempting to exchange the data. This is defined as part of the Common Data Definitions.
date	The *date* element is used by several elements. It is used to contain the actual date information. The structure of the *date* should conform to the IS8601 standard and takes the form of: YYYY-MM-DD i.e., year, month and day.
dateTime	The *dateTime* element is used by several elements. It is used to contain the actual date and/or time information. The structure of the *dateTime* should conform to the IS8601 standard and takes the form of: YYYY-MM-DDTHH:MM:SS i.e., year, month, day, hour, minute and second.
deleteGroup	This is the delete *Group* object behavior in the *Group Management Service*. The successful outcome is the deletion of the *Group* object on the *Provider.* If the supplied *SourcedId* cannot be located on the *Provider* then a failure status is returned.
deleteGroupRelationship	This is the delete a relationship from the *Group* object behavior in the *Group Management Service*. The successful outcome is the deletion of the relationship between the identified Group objects on the *Provider*; neither *Group* object is deleted. If an allocated *SourcedId* or *RelationId* cannot be located on the Provider then a failure status is returned.
deleteGroups	This is the delete many *Group* objects behavior in the *Group Management Service*. The successful outcome is the deletion of a set of populated *Group* objects on the *Provider.* If an allocated *SourcedId* cannot be located on the *Provider* then a failure status is returned for that part of the request.
deleteGroupsRelationship	This is the delete many relationships from *Group* objects behavior in the *Group Management Service*. The successful outcome is the deletion of a set of relationships from the populated *Group* objects on the *Provider*; no *Group* object is deleted If an allocated *SourcedId* cannot be located on the *Provider* then a failure status is returned for that part of the request. If an allocated *SourcedId* or *RelationId* cannot be located on the *Provider* then a failure status is returned for that part of the request.
deleteMembership	This is the delete *Membership* object behavior in the *Membership Management Service*. The successful outcome is the deletion of the *Membership* object on the *Provider.* If the supplied *SourcedId* cannot be located on the *Provider* then a failure status is returned.
deleteMemberships	This is the delete many *Membership* objects behavior in the *Membership Management Service*. The successful outcome is the deletion of a set of populated *Membership* objects on the *Service Provider.* If an allocated *SourcedId* cannot be located on the *Service Provider* then a failure status is returned for that part of the request.

deletePerson	This is the delete *Person* object behavior in the *Person Management Service*. The successful outcome is the deletion of the *Person* object on the *Service Provider*. If the supplied *SourcedId* cannot be located on the *Service Provider* then a failure status is returned.
deletePersons	This is the delete many *Person* objects behavior in the *Person Management Service*. The successful outcome is the deletion of a set of populated *Person* objects on the *Provider*. If an allocated *SourcedId* cannot be located on the *Service Provider* then a failure status is returned for that part of the request.
demographics	The *demographics* element is used to store the learner information for demographic information about the learner e.g., gender, place of birth, etc.
DemographicsDType	The *DemographicsDType* is an XSD complexType that is defined to act as the data type for demographics structures. The *demographics* element is of type *DemographicsDType*.
descFull	The *descFull* element is used within the *description* element. This element is used to contain a paragraph of text i.e., a string of characters (1-2048).
descLong	The *descLong* element is used within the *description* element. This element contains a long character string (i.e., 1-256characters) that is used to characterize the associated description material. The long entry is used in conjunction with the optional *descShort* and *descFull* elements.
description	The *description* element is used within the *group* element to describe the Group. The *description* element can include one or more of the *descShort*, *descLong* and *descFull* elements. The intention is for these three elements to contain related information about the same object i.e., to supply progressively increasing details.
DescriptionDType	The *DescriptionDType* is an XSD complexType that is defined to act as the data type for description structures. The *description* element is of type *DescriptionDType*.
descShort	The *descShort* element is used within the *description* element. This element is used to contain a short character string (i.e., less than 64 characters) that is used to characterize the associated description material. The short entry is used in conjunction with the optional *descLong* and *descFull* elements but every usage of the description element should have an associated *descShort* entry.
email	The *email* element is used in the *person* and *group* elements to store the corresponding email address for the Person or Group respectively. This is a character string [1-256]. This is defined as part of the Common Data Definitions.
end	The *end* element is used within the *timeframe* element to define when participation in a Group is to finish. It is a character string [1-20] containing the date in the ISO8601 format (YYYY-MM-DD).
enrollAccept	The *enrollaccept* element is used within the *enrollcontrol* element to indicate if enrolments to the Group are being accepted. The contents are enumerated as either ‘0=No’ or ‘1=Yes’ in the form of a single integer.

enrollAllowed	The *enrollallowed* element is used within the *enrollcontrol* element to indicate if the target system is permitted to enrol into the Group. The contents are enumerated as either ‘0=No’ or ‘1=Yes’ in the form of a single integer.
enrollControl	The *enrollcontrol* element is used within the *group* element to contain the enrolment conditions currently permitted for the Group. These enrolment conditions are defined using the *enrollaccept* and *enrollallowed* elements.
EnrollControlDType	The *EnrollControlDType* is an XSD complexType that is defined to act as the data type for enrollControl structures. The *enrollControl* element is of type *EnrollControlDType*.
extadd	The *extadd* element is used within the *address* element to provide extra address space. It is an optional field that can occur only once. It is used to contain any non-street components of the address e.g., suite number, company name, etc. It is a character string [1-128].
extension	This is the element used to contain the extension structures in the *Person*, *Group* and *Member* structures. It contains one or more *extensionField* elements.
ExtensionField	This element is the container for a single extension structure. It consists of the name of the extension field, *fieldName*, the data-type of the filed, *fieldType* and the content of the extension *fieldValue*. This is defined as part of the Common Data Definitions.
extRef	The *extRef* element is used within the *photo* element to contain the external reference for the image file. This reference could be a URL. It is a character string [1-1024].
fieldName	This is the UML attribute of the *ExtensionField* class that is used to hold the name of the extension field. It is a string of length 1-2048 characters. The equivalent element is defined as part of the Common Data Definitions.
fieldType	This is the UML attribute of the *ExtensionField* class that is used to hold the type of the extension field. It is a string of length 1-2048 characters. The type will have values such as ‘String’, ‘Boolean’ plus any of the other XSD scalar concrete data types. The equivalent element is defined as part of the Common Data Definitions.
fieldValue	This is the UML attribute of the *ExtensionField* class that is used to hold the value of the extension field. It is a string of length 1-2048 characters. The data must be handled as defined by the equivalent data type as described by the corresponding *fieldType* value. The equivalent element is defined as part of the Common Data Definitions.
finalResult	The *finalResult* element is used within the *role* element to store the final results assigned to the Person or Group. Each instantiation of the *finalResult* element is used to contain a single score.
firstId	This element is used to contain the first of two identifiers that are to be used as a couple. It is a character string [1-4096].
formatName	The *formatName* element is used within the *name* element. It is used to contain the formatted name as a character string [1-256]. The format of the contained name is undefined.

gender	The *gender* element is used within the *demographics* element to store the gender of the learner. It is an empty element and the information is entered through an attribute. The content is enumerated as ‘Unknown’, ‘Female’ and ‘Male’.
group	The *group* element is used to contain information about learning Groups. The Group construct is an abstract representation for any appropriate collection of learning activities/events e.g., classes, courses, etc. Group objects are manipulated using the *Group Management Service*.
Group Management Service	This is the service that is responsible for the management of *Group* objects. The service supports the manipulation of a single *Group* object, defined under the *GroupManager* interface, and a set of *Group* objects, defined under the *GroupsManager* interface.
GroupDType	The *GroupDType* is an XSD complexType that is defined to act as the data type for group structures. The *group* element is of type *GroupDType*.
groupIdPair	The *groupIdPair* is a tuple of the *sourcedId* and *Group* object itself (the data model for the object does not include a *sourcedId*). This is also an element that is used as a parameter in some of the behaviors of the *Group Management Service*.
GroupIdPairDType	The *GroupIdPairDType* is an XSD complexType that is defined to act as the data type for groupIdPair structures. The *groupIdPair* element is of type *GroupIdPairDType*.
groupIdPairSet	The *groupIdPairSet* is a collection of *groupIdPair* objects. This enables a set of *Group* objects with their *sourcedId* to be exchanged. This is also an element that is used as a parameter in some of the behaviors of the *Group Management Service*.
GroupIdPairSetDType	The *GroupIdPairSetDType* is an XSD complexType that is defined to act as the data type for groupIdPairSet structures. The *groupIdPairSet* element is of type *GroupIdPairSetDType*.
GroupManager	This is the interface class within the *Group Management Service* that contains the definition of the operations that control the management of a single *Group* object. The basic ‘CRUD’ operations are supported plus those that provide specialist control capabilities.
groupSet	The *groupSet* is a collection of *Group* objects. There is no identification of the contained *Group* objects i.e., there is no associated *sourcedId* for each object. This is also an element that is used as a parameter in some of the behaviors of the *Group Management Service*.
GroupSetDtype	The *GroupSetDType* is an XSD complexType that is defined to act as the data type for groupSet structures. The *groupSet* element is of type *GroupIdSetDType*.
GroupsManager	This is the interface class within the *Group Management Service* that contains the definition of the operations that control the management of a set of *Group* objects. The basic ‘CRUD’ operations are supported plus those that provide specialist control capabilities. There is an equivalent set operation for each of the individual *Group* object manipulation behaviors defined in the *GroupManager* interface.
groupSourcedId	This is a type of *sourcedId* derived from the *identifier* class. It is used to contain the *sourcedId* of a *Group*.

groupType	The *groupType* element is used within the *group* element to contain the information that allows the Group to be categorized into one or more coding schemes with any number of levels supported within each scheme.
GroupTypeDType	The *GroupTypeDType* is an XSD complexType that is defined to act as the data type for groupType structures. The *groupType* element is of type *GroupTypeDType*.
id	The *id* element is used within the *org* element to contain the identifier for that organization; this will not be a *sourcedId*. It is a character string [1-256].
identifier	The *identifier* class is the container for any form of global unique or locally unique identifier. The identifier is a flat string of length 1-4096 characters. The equivalent element is defined as part of the Common Data Definitions.
IdentifierDType	The *IdentifierDType* is an XSD complexType that is defined to act as the data type for identifier structures. The *identifier* element is of type *IdentifierDType*. This is defined as part of the Common Data Definitions.
identifierPair	This is a collection of two identifiers. The equivalent element is defined as part of the Common Data Definitions.
IdentifierPairDType	The *IdentifierPairDType* is an XSD complexType that is defined to act as the data type for identifierPair structures. The *identifierPair* element is of type *IdentifierPairDType*. This is defined as part of the Common Data Definitions.
identifierPairSet	This is a set of *identifierPair*s. There is no significance in the order of the identifiers. The equivalent element is defined as part of the Common Data Definitions. This is defined as part of the Common Data Definitions.
IdentifierPairSetDType	The *IdentifierPairSetDType* is an XSD complexType that is defined to act as the data type for identifierPair structures. The *identifierPairSet* element is of type *IdentifierPairSetDType*. This is defined as part of the Common Data Definitions.
identifierSet	This is a set of *identifier*s. There is no significance in the order of the identifiers. The equivalent element is defined as part of the Common Data Definitions.
IdentifierSetDType	The *IdentifierSetDType* is an XSD complexType that is defined to act as the data type for identifierPair structures. The *identifierSet* element is of type *IdentifierSetDType*. This is defined as part of the Common Data Definitions.
idType	The *idType* element is used within the *member* element to indicate if the Member is a Group or a Person. The content is enumerated as either ‘1=Person’ or ‘2=Group’. It is an integer [1 or 2].
imgType	The *imgType* element is used with the *photo* element to identify the type of image. It is recommended that MIME notation be used. This is a character string [1-32].
IMSextension	This is the class that is defined to be the generic extension facility. This is defined as part of the Common Data Definitions.

IMSExtensionDType	The *IMSExtensionDType* is an XSD complexType that is defined to act as the data type for extension structures. The *extension* element is of type *IMSExtensiontDType*. The equivalent element is defined as part of the Common Data Definitions.
institutionRole	The *institutionRole* element is used within the *person* element to contain the roles of the individual within the institution. Each role is described using a separate instantiation of the element. This is an empty element with the data selected from the enumerated vocabulary contained within the *institutionRoleType* attribute. The *primaryRole* attribute is used to indicate if the corresponding role is the Person’s primary role within the institution.
InstitutionRoleDType	The *InstitutionRoleDType* is an XSD complexType that is defined to act as the data type for institution role structures. The *institutionRole* element is of type *InstitutionDType*.
institutionRoleType	The *institutionRoleType* is a mandatory element for the *institutionRole* element. It is a character string [1-32] that has the enumerated content of: Student, Faculty, Member, Learner, Instructor, Mentor, Staff, Alumni, ProspectiveStudent, Guest, Other, Administrator and Observer.
interimResult	The *interimResult* element is used within the *role* element to store the interim results assigned to the Person or Group. Each instantiation of the *interimResult* element is used to contain a single score. Multiple interim results can be returned and the *resultType* attribute is used to define the type of interim result e.g., Mid-term, etc.
label	The *label* element is used within the *relationship* element to describe the nature of the relationship between the host Group and the Group being described. It is a short human-readable description. This is a character string [1-32].
LangString	This class enables the enclosed text to have its associated language identified. It contains the *language* and text attributes. This is defined as part of the Common Data Definitions.
LangStringDType	The *LangStringDType* is an XSD complexType that is defined to act as the data type for language structures. The *langString* element is of type *LangStringDType*. This is defined as part of the Common Data Definitions.
language	This is the attribute that contains the type of language identifier in the *LangString* class. The value should be as defined in ISO603.
level	The *level* element is used with the *typevalue* element to indicate the coding level assigned to the group. Level ‘1’ is usually the highest level with level ‘2’ a further refinement, etc. This is a character string [1-2].
list	The *list* element(s) is used within the *values* element to contain the specific value that the associated result may take. This element is only used if the *valueType* element of the *values* element is equal to zero (0). If the *list* element is used then the value recorded in the associated *results* element must be contained within one of the instantiations of *list*. It is a character string [1-32].

locality	The *locality* element is used within the *address* element to store the locality part of an address. The locality of an address refers to the immediate geographic area around the street or complex. This could be the parish, the county, etc. The data is entered as a string of up to 64 characters.
max	The *max* element is used within the *values* element to define the maximum numerical value that can be assigned to the interim or final result. This element is only used if the *valueType* attribute of the *values* element is set to one (1). It is a decimal number in the range 0-9999.9999.
member	The *member* element is used within the *membership* element to contain the details about the Group's or Person's role within the host membership Group. At least one set of member details must be defined within the membership element.
MemberDType	The *MemberDType* is an XSD complexType that is defined to act as the data type for member structures. The *member* element is of type *MemberDType*.
membership	The *membership* element is within the root element to contain information about the members of a *Group*. The members may be a *Person* or another *Group*.
Membership Management Service	This is the service that is responsible for the management of *Membership* objects. The service supports the manipulation of a single *Membership* object, defined under the *MembershipManager* interface, and a set of *Membership* objects, defined under the *MembershipsManager* interface.
MembershipDType	The *MembershipDType* is an XSD complexType that is defined to act as the data type for membership structures. The *membership* element is of type *MembershipDType*.
membershipIdPair	The *membershipIdPair* is a tuple of the *sourcedId* and *Membership* object itself (the data model for the object does not include a *sourcedId*). This is also an element that is used as a parameter in some of the behaviors of the *Membership Management Service*.
MembershipIdPairDType	The *MembershipIdPairDType* is an XSD complexType that is defined to act as the data type for membership/identifier pair structures. The *membershipIdPair* element is of type *MembershipIdPairDType*.
membershipIdPairSet	The *MembershipIdPairSet* is a collection of *MembershipIdpair* objects. This enables a set of *Membership* objects with their *sourcedId* to be exchanged. This is also an element that is used as a parameter in some of the behaviors of the *Membership Management Service*.
MembershipIdPairSetDType	The *MembershipIdPairSetDType* is an XSD complexType that is defined to act as the data type for a set of membership/identifier pair structures. The *membershipIdPairSet* element is of type *MembershipIdPairSetDType*.
MembershipManager	This is the interface class within the *Membership Management Service* that contains the definition of the operations that control the management of a single *Membership* object. The basic 'CRUD' operations are supported plus those that provide specialist control capabilities.

membershipSet	The *membershipSet* is a collection of *Membership* objects. There is no identification of the contained *Membership* objects i.e., there is no associated *sourcedId* for each object. This is also an element that is used as a parameter in some of the behaviors of the *Membership Management Service*.
MembershipSetDType	The *MembershipSetDType* is an XSD complexType that is defined to act as the data type for a set of membership structures. The *membershipSet* element is of type *MembershipSetDType*.
MembershipsManager	This is the interface class within the *Membership Management Service* that contains the definition of the operations that control the management of a set of *Membership* objects. The basic 'CRUD' operations are supported plus those that provide specialist control capabilities. There is an equivalent set operation for each of the individual *Membership* object manipulation behaviors defined in the *MembershipManager* interface.
memberSourcedId	This is a type of *sourcedId* derived from the *identifier* class. It is used to contain the *sourcedId* of a *Member.*
messageRefIdentifier	This element is used within the *statusInfo* element to contain the message identifier of the request message. This element should only be used in response messages to enable the *Service Requester* to link the response with the original request. This is character string [1-32].
messageIdentifier	This element is used within the request and response messages to contain the unique message identifier allocated to the message. This element is contained within the *syncRequestHeaderInfo* and *syncResponseHeaderInfo* elements.
min	The *min* element is used within the *values* element to define the minimum numerical value that can be assigned to the interim or final result. This element is only used if the *valueType* attribute of the *values* element is set to one (1). It is a decimal number in the range 0-9999.9999.
mode	The *mode* element is an optional element within the *finalResult* and *interimResult* elements to provide a short descriptive name for the type of scoring supplied e.g., 'Letter Grade', 'Percentage', etc. It is a character string [1-32].
name	The *name* element is used within the *person* element. It is used to store the appropriate name of the learner. Each name is exchanged using its own *name* element. The name is entered through its component parts i.e., using the *partName* element. The type of name is identified using the *nameType* element.
NameDType	The *NameDType* is an XSD complexType that is defined to act as the data type for name structures. The *name* element is of type *NameDType*.
namePartType	The *namePartType* element is used to contain the type of name part being supplied. This is an open vocabulary and will include string values such as 'First', 'Last', 'Surname', 'Other', etc. This is a character string of length 1-32.
namePartValue	The *namePartValue* element is used to contain the name part being supplied. This is a character string of length 1-256.

nameType	The *nameType* element is used to contain the type of name part being supplied. This is an open vocabulary and will include string values such as ‘Full’, ‘Pseudonym’, ‘Other’, etc. This is a character string of length 1-32.
operationRefIdentifier	This element is used within the *statusInfo* to contain information that can be used by the *System Provider* to examine the information returned by the *Service Requester.* The manner in which this information is used is implementation dependent. This is a character string [1-32].
org	The *org* element is used within the *group* element to identify the organization that is administering or sponsoring the Group.
OrgDType	The *OrgDType* is an XSD complexType that is defined to act as the data type for organization structures. The *org* element is of type *OrgDType*.
orgName	The *orgName* element is used within the *org* element to store the name of the sponsoring or administering organization. It is a character string [1-256].
orgUnit	The *orgUnit* element is used within the *org* element to contain the name of the sponsoring or administering unit within the organization. It is a character string [1-256]. Multiple units within the organization can sponsor or administer the same Group.
pairSourcedId	This element is used to contain a pair of *sourcedId*s. The nature of the relationship is undefined.
pairSourcedIdSet	This is an unordered set of *pairSourcedId*s.
partName	The *partName* element is used within the *name* element to contain a component of a name. The type of component is identified using the *namePartType* element and the assigned name part is contained in the element *namePartValue*.
passWord	The *password* element is the part of the *userid* element is used to contain the password allocated to the user. It is a character string [1-1024]. This is the actual password for the user and so this should be encrypted using the algorithm identified in then associated *pwEncryptionType* element.
person	The *person* data model is the root data structure for the *Person Management Service*. It is also defined as an element that is used in some of the behaviors in the *Person Management Service*.
Person Management Service	This is the service that is responsible for the management of *Person* objects. The service supports the manipulation of a single *Person* object, defined under the *PersonManager* interface, and a set of *Person* objects, defined under the *PersonsManager* interface.
PersonDType	The *PersonDType* is an XSD complexType that is defined to act as the data type for person structures. The *person* element is of type *PersonDType*.
personIdPair	The *personIdPair* is a tuple of the *sourcedId* and *Person* object itself (the data model for the object does not include a *sourcedId*). This is also an element that is used as a parameter in some of the behaviors of the *Person Management Service*.

personIdPairSet	The *personIdPairSet* is a collection of *personIdpair* objects. This enables a set of *person* objects with their *sourcedId* to be exchanged. This is also an element that is used as a parameter in some of the behaviors of the *Person Management Service*.
PersonManager	This is the interface class within the *Person Management Service* that contains the definition of the operations that control the management of a set of *Person* objects. The basic ‘CRUD’ operations are supported plus those that provide specialist control capabilities. There is an equivalent set operation for each of the individual *Person* object manipulation behaviors defined in the *PersonManager* interface.
personSet	The *personSet* is a collection of *Person* objects. There is no identification of the contained *Person* objects i.e., there is no associated *sourcedId* for each object. This is also an element that is used as a parameter in some of the behaviors of the *Person Management Service*.
PersonsManager	This is the interface class within the *Person Management Service* that contains the definition of the operations that control the management of a set of *Person* objects. The basic ‘CRUD’ operations are supported plus those that provide specialist control capabilities. There is an equivalent set operation for each of the individual *Person* object manipulation behaviors defined in the *PersonManager* interface.
photo	The *photo* element is used within the *person* element. It is used to contain the reference to an external location where the Person’s photograph is stored. Each Person structure may contain one *photo* element.
PhotoDtype	The *PhotoDType* is an XSD complexType that is defined to act as the data type for photograph structures. The *photo* element is of type *PhotoDType*.
pobox	The *pobox* element is used within the *address* element within the *person* element. It is used to store the PO Box number component of the address – if available. It is a character string [1-32].
postcode	The *postcode* element is used within the *address* element only. This element is used to contain the post-code component of the Person’s address. It is a character string [1-32]. The format of the code will vary from country to country.
primaryRoleType	The *primaryRoleType* attribute is used on the *institutionRole* element to indicate if the associated role is the primary role within the institution. This is a mandatory attribute and it content is enumerated as ‘Yes’ (it is the primary role) and ‘No’ it is not the primary role.
pwEncryptionType	The *pwEncryptionType* attribute is used on the *userid* element to describe the type of encryption used on the password (as carried in the *password* attribute). Possible content includes ‘PKC’, ‘MD5’, etc. It is a character string [1-32].
readGroup	This is the read *Group* object behavior in the *Group Management Service*. The successful outcome is that the *Group* object identified by the *Service Requester* is returned by the *Service Provider.* If the supplied *SourcedId* cannot be located on the *Service Provider* then a failure status is returned.

readGroups	This is the read many *Group* objects behavior in the *Group Management Service*. The successful outcome is that the set of *Group* objects identified by the *Service Requester* are returned by the *Service Provider.* If an allocated *SourcedId* cannot be located on the *Service Provider* then a failure status is returned for that part of the request.
readGroupsForPerson	This is the operation in the *Group Management Service* to request the set of *Group* objects associated with the defined *Person*. This operation is defined under the *GroupsManager* interface. If the *SourcedId* for the *Person* cannot be located on the *Service Provider* then an error code is returned. The status information is returned as a *StatusInfoSet* object. All of the group objects for which the person is a member are returned.
readMembership	This is the read *Membership* object behavior in the *Membership Management Service*. The successful outcome is that the *Membership* object identified by the *Service Requester* is returned by the *Service Provider.* If the supplied *SourcedId* cannot be located on the *Service Provider* then a failure status is returned.
readMemberships	This is the read many *Membership* objects behavior in the *Person Management Service*. The successful outcome is that the set of *Membership* objects identified by the *Service Requester* are returned by the *Service Provider.* If an allocated *SourcedId* cannot be located on the *Service Provider* then a failure status is returned for that part of the request.
readMembershipsForGroup	This is the operation in the *Membership Management Service* to request the set of *Membership* objects associated with the defined *Group*. This operation is defined under the *MembershipsManager* interface. If the *SourcedId* for the *Group* cannot be located on the *Service Provider* then an error code is returned. The status information is returned as a *StatusInfoSet* object. All of the membership objects for the group are returned.
readMembershipsForPerson	This is the operation in the *Membership Management Service* to request the set of *Membership* objects associated with the defined *Person*. This operation is defined under the *MembershipsManager* interface. If the *SourcedId* for the *Person* cannot be located on the *Service Provider* then an error code is returned. The status information is returned as a *StatusInfoSet* object. All of the membership objects for the person are returned.
readPerson	This is the read *Person* object behavior in the *Person Management Service*. The successful outcome is that the *Person* object identified by the *Service Requester* is returned by the *Service Provider.* If the supplied *SourcedId* cannot be located on the *Service Provider* then a failure status is returned.
readPersons	This is the read many *Person* objects behavior in the *Person Management Service*. The successful outcome is that the set of *Person* objects identified by the *Service Requester* are returned by the *Service Provider.* If an allocated *SourcedId* cannot be located on the *Service Provider* then a failure status is returned for that part of the request.

readPersonsForGroup	This is the operation in the *Person Management Service* to request the set of *Person* objects associated with the defined *Group*. This operation is defined under the *PersonsManager* interface. If the *SourcedId* for the *Group* cannot be located on the *Service Provider* then an error code is returned. The status information is returned as a *StatusInfoSet* object. All of the person objects for that group are returned.
recordInfo	This is the element that is sued within the *Person*, *Group* and *Role* data objects to contain the meta-data information about those objects. This is an object instance of the *RecordMetaData* class.
RecordMetaData	This is the class that is the container for meta-data on the associated data structure. This is defined as part of the Common Data Definitions.
RecordMetaDataDType	The *RecordMetaDataDType* is an XSD complexType that is defined to act as the data type for record meta-data/comment structures. The *recordMetaData* element is of type *RcordMetaDataDType*.
region	The *region* element is used within the *address* element only. This element is used to contain the region parts of an address e.g., ‘Europe’, ‘South America’, etc. An address may or may not contain an associated region part. The information is entered as a string of up to 64 characters.
relation	The *relation* element is used with the *relationship* element to define the nature of the relationship. The content is enumerated as ‘1=Parent’, ‘2=Child’ and ‘3=Also known as’, ‘Parent’, ‘Child’ and ‘KnownAs’.
relationId	This is a parameter used in the *deleteGroupRelationship* behavior. It is used to identify the relationship being deleted. The identifier of the *relationship* was assigned by the *sourcedId* part of the relationship element as part of the *Group* data structure management.
relationship	The *relationship* element is used within the *group* element to define the relationship of the host group and the group defined by the relationship. The type of relationship is identified using the *relation* attribute. The relationship is expressed as ‘A’ is the ‘relation’ of ‘B’ where ‘A’ is the Group identified within the *relationship* element (the contained *sourcedid* element) and ‘B’ is the Group containing the *relationship* element.
RelationshipDType	The *RelationshipDType* is an XSD complexType that is defined to act as the data type for relationship structures. The *relationship* element is of type *RelationshipDType*.
restrict	The *restrict* element is used by the *begin* and *end* elements to contain the nature of the restriction. The content is enumerated as ‘0=No’ or ‘1=Yes’. It is a boolean.
RestrictDate	The *restrictDate* is a class used for the *begin* and *end* attributes of the *TimeFrame* class to define if learner participation is permitted before the begin date or after the end data. Its two attributes are the *date* for the restriction and the *restrict* nature. This is defined as part of the Common Data Definitions.
RestrictDateDType	The *RestrictDateDType* is an XSD complexType that is defined to act as the data type for restricted date structures. The *restrictDate* element is of type *RestrictDateDType*. This is defined as part of the Common Data Definitions.

result	The *result* element is used within the *interimResult* and *finalResult* elements to contain the actual result being exchanged (only one value can be exchanged per instantiation of the *interimResult* and *finalResult* element). The type of the result being exchanged is defined by the *values* element also contained within the corresponding *interimResult* and *finalResult* element. It is a character string [1-32].
ResultDtype	The *ResultDType* is an XSD complexType that is defined to act as the data type for result structures. The *result* element is of type *ResultDType*.
resultType	The *resultType* is an optional element is used on the *interimResult* and *finalResult* elements to define the type of interim result being reported e.g., ‘Mid-term’, ‘Part I’, etc. There is no predefined vocabulary for this content.
role	The *role* element is used within the *member* element to contain the description of the role that the Member has as a part of its membership. The *member* element must contain at least one role description and a Member can have more than one role. The type of role is defined using the *roleType* element.
RoleDType	The *RoleDType* is an XSD complexType that is defined to act as the data type for role information structures. The *role* element is of type *RoleDType*.
roleType	The *roletype* attribute is used with the *role* element to contain the role type definition. The content is enumerated as ‘01=Learner’, ‘02=Instructor’, ‘03=Content developer’, ‘04=Member’, ‘05=Manager’, ‘06=Mentor’, ‘07=Administrator’, ‘08=Teaching Assistant’, ‘Learner’, ‘Instructor’, ‘Content developer’, ‘Member’, ‘Manager’, ‘Mentor’, ‘Administrator’, ‘Teaching Assistant’.
scheme	The *scheme* element is used with the *groupType* element to contain the coding scheme. There is no agreed vocabulary for the coding scheme and so interoperability will require the agreement of an appropriate vocabulary as part of the business process mapping. It is a character string [1-256].
secondId	This element is used to contain the second of two identifiers that are to be used as a couple. It is a character string [1-4096].
Service Provider	The Service Provider is the system that responds to the requests made by a *Service Requester.* The Service Provider is the provider of the service being requested. The Service Provider creates and sends the response messages.
Service Requester	The Service Requester is the system that makes a service request. The request is sent to a *Service Provider* using the appropriate implementation. The Service Requester creates and sends the request messages.
severity	This is an attribute of the *StatusInfo* class and an element within the *statusInfo* element. It is used to contain the description of the severity of the status information being reported. It consists of an enumerated set of ‘Status’, ‘Warning’ and ‘Error’. This is defined as part of the Common Data Definitions.

sourcedId	The *sourcedId* is a unique identifier assigned to an object with the *Enterprise Service*. It is a type of *identifier* and as such it is a flat string. This new *sourcedId* can contain the sourcedId defined in the Enterprise Specification v1.1 by concatenating the original source and id elements and using an appropriate string delimiter.
sourcedIdSet	This is a set of *sourcedId*s. The order within the set may be significant.
status	The *status* element is used, mandatory, within the *role* element to indicate if the Member is active or inactive within the Group. The content is enumerated as ‘0=Inactive’ or ‘1=Active’. This is a Boolean.
statusInfo	This element is the container for all of the status information being reported. This information is passed as part of the response message header i.e., in the *syncResponseHeaderInfo* element. This is defined as part of the Common Data Definitions.
StatusInfoDType	The *StatusInfoDType* is an XSD complexType that is defined to act as the data type for status information structures. The *statusInfo* element is of type *StatusInfoDType*. This is defined as part of the Common Data Definitions.
statusInfoSet	This is the container for a set of *statusInfo* reports. This element is used if the message contains a report for a behavior that requested activity on two or more data objects. This is defined as part of the Common Data Definitions.
StatusInfoSetDType	The *StatusInfoSetDType* is an XSD complexType that is defined to act as the data type for a set of status information structures. The *statusInfoSet* element is of type *StatusInfoSetDType*. This is defined as part of the Common Data Definitions.
street	The *street* element is used within the *adr* element. It is used to contain the details of the street part of the full address. An address may or may not contain information about the street e.g., this may be unnecessary if a PO Box number has given. This is a three line character string each of length 1-128.
subRole	The *subRole* element is used within the *role* element to provide further qualification on the member’s role within the Group. There is no agreed vocabulary to describe the various possible subroles. It is a character string [1-32].
syncRequestHeaderInfo	This element is the container of the header information for the request messages exchanged as part of a synchronous communications exchange. It contains the unique message identifier. This is defined as part of the Common Data Definitions.
syncResponseHeaderInfo	This element is the container of the header information for the response messages exchanged as part of a synchronous communications exchange. It contains the unique message identifier, the status information for the request and the message identifier of the original request message. This is defined as part of the Common Data Definitions.
systemRole	The *systemRole* element is used within the *person* element to contain the role of the individual with the computer system environment. The content is enumerated as ‘SysAdmin’, ‘SysSupport’, ‘Creator’, ‘AccountAdmin’, ‘User’, ‘Administrator’ or ‘None’.

tel	The *tel* element is used within the *person* element. This element is used to store the appropriate telephone number (including mobile telephone, pager and facsimile numbers). The telephone number can include the county code and extension number as well as the normal area code and actual number.
TelDType	The *telDType* is an XSD complexType that is defined to act as the data type for telephone structures. The *tel* element is of type *TelDType*.
telType	The *telType* attribute is used with the *tel* element to define the type of telephone number being stored. The content is enumerated as ‘1=Voice’, ‘2=Fax’, ‘3=Mobile’, ‘4=Pager’, ‘Voice’, ‘Fax’, Mobile’ and ‘Pager’.
telValue	This is the container for the telephone number element *tel*. It is a character string [1-32].
text	This is an attribute of the *LangString* class. It is used to contain the text string. This is a character string [1-2048].
timeFrame	The *timeFrame* element is used within the *group* and *role* elements to define the period during which the Group or the Membership is active. The period is defined using the *begin* and *end* elements plus the *adminPeriod* element that is used to supply a short human-readable description of the period. The equivalent class is defined as part of the Common Data Definitions.
TimeFrameDType	The *TimeFrameDType* is an XSD complexType that is defined to act as the data type for time-frame structures. The *timeFrame* element is of type *TimeFrameDType*. This is defined as part of the Common Data Definitions.
type	This element is used within the *groupType* element to contain the actual type of Group. There is no agreed vocabulary. It is a character string [1-256].
typeValue	The *typeValue* element is used within the *groupType* element to contain the classification assigned to the associated Group. Each Group must have at least one *typeValue* classification if its group type has been defined. It is a character string [1-256].
TypeValueDType	The *TypeValueDType* is an XSD complexType that is defined to act as the data type for typeValue structures. The *typeValue* element is of type *TypeValueDType*.
updateGroup	This is the update a single *Group* object behavior in the *Group Management Service*. The update behavior is additive. In the case of a multiple occurring field a new field is added otherwise the current content is replaced.
updateGroups	This is the update a set of *Group* objects behavior in the *Group Management Service*. The update behavior is additive. In the case of a multiple occurring field a new field is added otherwise the current content is replaced. A *Provider* will complete as many of the update request as possible and will provide a status report on each attempted update.
updateMembership	This is the update a single *Membership* object behavior in the *Membership Management Service*. The update behavior is additive. In the case of a multiple occurring field a new field is added otherwise the current content is replaced.

updateMemberships	This is the update a set of *Membership* objects behavior in the *Membership Management Service*. The update behavior is additive. In the case of a multiple occurring field a new field is added otherwise the current content is replaced. A *Provider* will complete as many of the update request as possible and will provide a status report on each attempted update.
updatePerson	This is the update a single *Person* object behavior in the *Person Management Service*. The update behavior is additive. In the case of a multiple occurring field a new field is added otherwise the current content is replaced.
updatePersons	This is the update a set of *Person* objects behavior in the *Person Management Service*. The update behavior is additive. In the case of a multiple occurring field a new field is added otherwise the current content is replaced. A *Provider* will complete as many of the update request as possible and will provide a status report on each attempted update.
url	The *url* element is used to contain the URL for the Person or Group. This is a character string [1-4096] that contains the absolute form of the URL. This is defined as part of the Common Data Definitions.
userId	The *userId* element is used within the *person* and *role* elements to contain the user’s identification information. It uses the *password*, *pwEncryptionType* and *authenticationType* attributes to contain information that supports the storage of the user’s identification information. The type of user identifier is contained in the element *userIdType* and the value in the element *userIdValue*.
UserIdDType	The *UserIdDType* is an XSD complexType that is defined to act as the data type for user identification structures. The *userId* element is of type *UserIdDType*. This is defined as part of the Common Data Definitions.
userIdType	The *userIdType* attribute is used on the *userId* element to identify the type of user identification. Some examples of this are ‘NationalId’ or ‘InstitutionId’. There is no defined vocabulary for the range of possible identification entries. It is a character string [1-32]. This is defined as part of the Common Data Definitions.
userIdValue	This element is the container of the user identification defined as *userId*. It is a character string [1-256]. This is defined as part of the Common Data Definitions.
values	The *values* element is used within the *finalResult* and *interimResult* elements to contain the description of the values that the result element may contain e.g., a grade from a list of possible grades or a numeric value between some minimum and maximum.
ValuesDType	The *ValuesDType* is an XSD complexType that is defined to act as the data type for values structures. The *values* element is of type *ValuesDType*.
valueType	The *valueType* element is used, mandatory, on the *values* element to indicate the type of values being described. The content is enumerated as ‘0=List’ and ‘1=Range’. The range is defined by a numeric minimum and maximum and the list is defined through a list of the possible grades.

Appendix B – OKI Enterprise Services OSIDs

The core deliverable of the Open Knowledge Initiative (OKI) is an architectural specification in support of learning management and educational application development, the primary feature of which is a set of application programming interface (API) definitions. OKI are specifying the architecture by identifying a set of services, a framework, upon which learning tool developers can base their work. One of the Educational Services defined under OKI is the Course Management Open Service Interface Definition [OKI, 03].

The CourseManagement Open Service Interface Definition (OSID) primarily supports creating and managing a CourseCatalog. The catalog is organized into:

- CanonicalCourses, which are general and exist across terms;
- CourseOfferings for CanonicalCourses, which occur in a specific term, have a GradeType, a StatusType, etc.;
- CourseSections for CourseOfferings, which have a meeting location, schedule, student roster, etc.

When used in concert, the OSIDs comprise a complete system with each Service focused exclusively on a particular area. For example, the roles related to a CourseOffering and CourseSection are defined through the Authorization OSID; coursework and course material can be defined in the Assessment and DigitalRepository OSIDs; course grades are assigned through the Grading OSID, and so on.

An examination of an Open Service Interface Definition usually begins with the Manager. All Managers provide the way to create the objects that implement the principal interfaces in the Service. Before discussing the Manager in detail, we will review the intended function of the CourseManagement Service overall. The top-tier organizing structure for CourseManagement is the CanonicalCourse. A CanonicalCourse includes several properties: a title, a description, a number, a unique Id, a type, and number of credits. The purpose of these properties is to provide enough structure and flexibility to map a CanonicalCourse to the various institutions' systems for describing academic courses. CanonicalCourse also include:

- A list of any equivalent CanonicalCourses (for cross-listing);
- A list of any prerequisites;
- A list of any CanonicalCourses (hierarchical children);
- A list of any CourseOfferings.

CanonicalCourses are designed for use across many Terms and might be archived, even after there are no longer CourseOfferings for it. CanonicalCourses are not intended to capture information about the Course for a specific Term. That is the role of the CourseOffering.

The CourseOffering contains the same title, Id, number, and description properties as a CanonicalCourse. These properties' values can be the same as those for the CanonicalCourse or overridden for this particular CourseOffering. In place of the CanonicalCourse's CourseType there is an OfferingType. The CourseOffering does not include a credits property but it does include a GradeType. The CourseManagementManager supports creating, deleting, and getting grades of this Type for any Agent and CourseOffering. The CourseOffering is specific to a Term.

CourseOfferings include an AssetId property. This is a place to put a reference to some DigitalRepository or other Asset associated with the CourseOffering. CourseOfferings also include a StatusType. This could be used to indicate open, closed, discontinued, etc. but as with all Types, there are no restrictions on the meaning of the Type. The CourseOffering includes both a list of the parent CanonicalCourses (there must be at least one) as well as any created CourseSections.

The CourseSection again contains title, description, number and Id properties which can be those of the parent CourseOffering or overridden. The CourseSection is in what Students actually enroll and it has a Schedule, a location, and manages a roster. As with the CourseOffering, there is a property for a Asset reference.

The UML visualization of the OKI CourseManagement OSID is shown in Figure B.1.

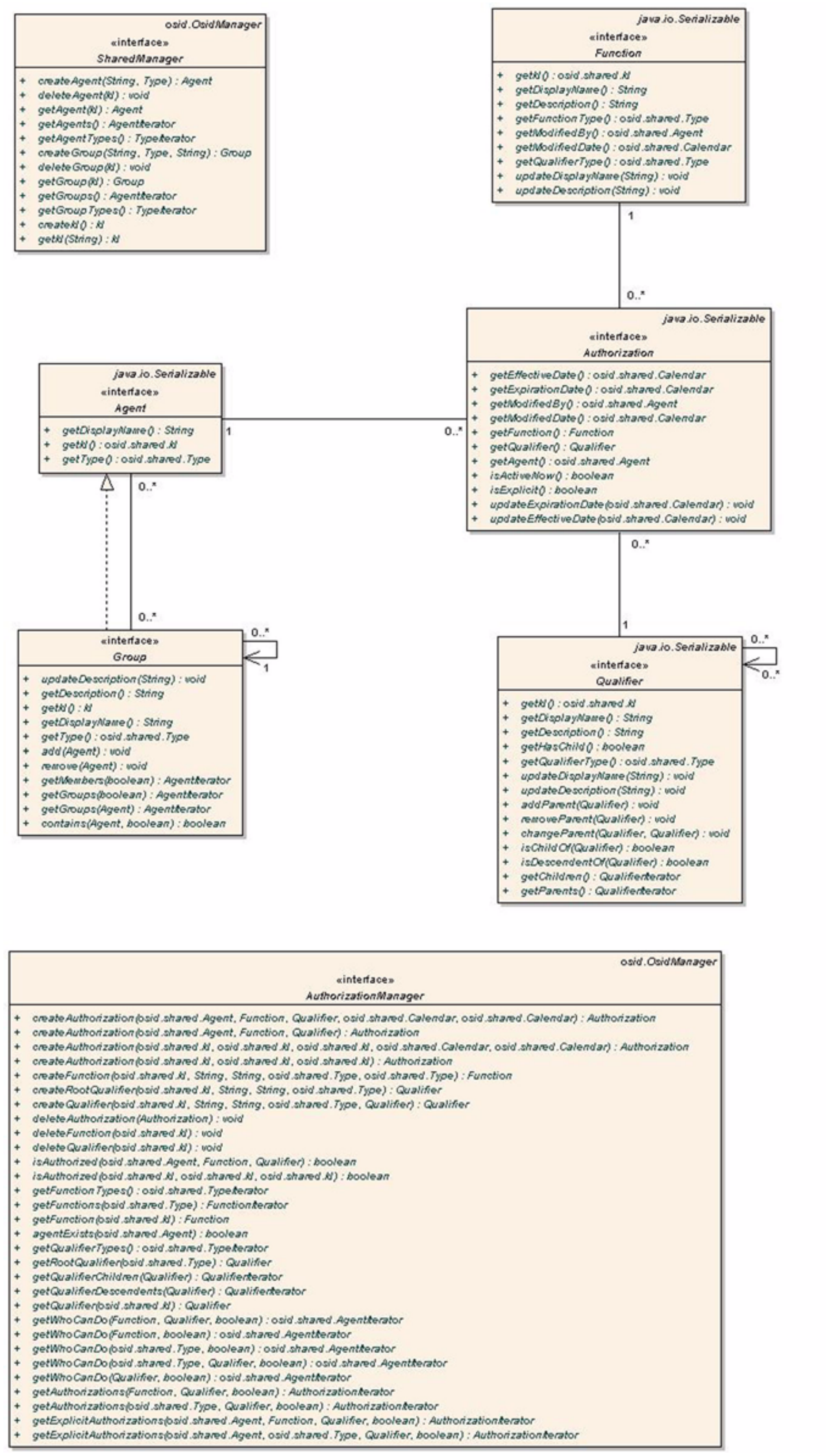

osid.OsidManager
«interface»
SharedManager
+ createAgent(String, Type) : Agent
+ deleteAgent(Id) : void
+ getAgent(Id) : Agent
+ getAgents() : AgentIterator
+ getAgentTypes() : TypeIterator
+ createGroup(String, Type, String) : Group
+ deleteGroup(Id) : void
+ getGroup(Id) : Group
+ getGroups() : AgentIterator
+ getGroupTypes() : TypeIterator
+ createId() : Id
+ getId(String) : Id
java.io.Serializable
«interface»
Function
+ getId() : osid.shared.Id
+ getDisplayName() : String
+ getDescription() : String
+ getFunctionType() : osid.shared.Type
+ getModifiedBy() : osid.shared.Agent
+ getModifiedDate() : osid.shared.Calendar
+ getQualifierType() : osid.shared.Type
+ updateDisplayName(String) : void
+ updateDescription(String) : void
1
0..*
java.io.Serializable
«interface»
Authorization
+ getEffectiveDate() : osid.shared.Calendar
+ getExpirationDate() : osid.shared.Calendar
+ getModifiedBy() : osid.shared.Agent
+ getModifiedDate() : osid.shared.Calendar
+ getFunction() : Function
+ getQualifier() : Qualifier
+ getAgent() : osid.shared.Agent
+ isActiveNow() : boolean
+ isExplicit() : boolean
+ updateExpirationDate(osid.shared.Calendar) : void
+ updateEffectiveDate(osid.shared.Calendar) : void
java.io.Serializable
«interface»
Agent
+ getDisplayName() : String
+ getId() : osid.shared.Id
+ getType() : osid.shared.Type
1
0..*
0..*
0..*
0..*
1
«interface»
Group
+ updateDescription(String) : void
+ getDescription() : String
+ getId() : Id
+ getDisplayName() : String
+ getType() : osid.shared.Type
+ add(Agent) : void
+ remove(Agent) : void
+ getMembers(boolean) : AgentIterator
+ getGroups(boolean) : AgentIterator
+ getGroups(Agent) : AgentIterator
+ contains(Agent, boolean) : boolean
0..*
1
java.io.Serializable
«interface»
Qualifier
+ getId() : osid.shared.Id
+ getDisplayName() : String
+ getDescription() : String
+ getHasChild() : boolean
+ getQualifierType() : osid.shared.Type
+ updateDisplayName(String) : void
+ updateDescription(String) : void
+ addParent(Qualifier) : void
+ removeParent(Qualifier) : void
+ changeParent(Qualifier, Qualifier) : void
+ isChildOf(Qualifier) : boolean
+ isDescendentOf(Qualifier) : boolean
+ getChildren() : QualifierIterator
+ getParents() : QualifierIterator
0..*
0..*
osid.OsidManager
«interface»
AuthorizationManager
+ createAuthorization(osid.shared.Agent, Function, Qualifier, osid.shared.Calendar, osid.shared.Calendar) : Authorization
+ createAuthorization(osid.shared.Agent, Function, Qualifier) : Authorization
+ createAuthorization(osid.shared.Id, osid.shared.Id, osid.shared.Id, osid.shared.Calendar, osid.shared.Calendar) : Authorization
+ createAuthorization(osid.shared.Id, osid.shared.Id, osid.shared.Id) : Authorization
+ createFunction(osid.shared.Id, String, String, osid.shared.Type, osid.shared.Type) : Function
+ createRootQualifier(osid.shared.Id, String, String, osid.shared.Type) : Qualifier
+ createQualifier(osid.shared.Id, String, String, osid.shared.Type, Qualifier) : Qualifier
+ deleteAuthorization(Authorization) : void
+ deleteFunction(osid.shared.Id) : void
+ deleteQualifier(osid.shared.Id) : void
+ isAuthorized(osid.shared.Agent, Function, Qualifier) : boolean
+ isAuthorized(osid.shared.Id, osid.shared.Id, osid.shared.Id) : boolean
+ getFunctionTypes() : osid.shared.TypeIterator
+ getFunctions(osid.shared.Type) : FunctionIterator
+ getFunction(osid.shared.Id) : Function
+ agentExists(osid.shared.Agent) : boolean
+ getQualifierTypes() : osid.shared.TypeIterator
+ getRootQualifier(osid.shared.Type) : Qualifier
+ getQualifierChildren(Qualifier) : QualifierIterator
+ getQualifierDescendents(Qualifier) : QualifierIterator
+ getQualifier(osid.shared.Id) : Qualifier
+ getWhoCanDo(Function, Qualifier, boolean) : osid.shared.AgentIterator
+ getWhoCanDo(Function, boolean) : osid.shared.AgentIterator
+ getWhoCanDo(osid.shared.Type, boolean) : osid.shared.AgentIterator
+ getWhoCanDo(osid.shared.Type, Qualifier, boolean) : osid.shared.AgentIterator
+ getWhoCanDo(Qualifier, boolean) : osid.shared.AgentIterator
+ getAuthorizations(Function, Qualifier, boolean) : AuthorizationIterator
+ getAuthorizations(osid.shared.Type, Qualifier, boolean) : AuthorizationIterator
+ getExplicitAuthorizations(osid.shared.Agent, Function, Qualifier, boolean) : AuthorizationIterator
+ getExplicitAuthorizations(osid.shared.Agent, osid.shared.Type, Qualifier, boolean) : AuthorizationIterator

Figure B.1 The OKI equivalent Enterprise Services OSIDs.

About This Document

Title	IMS Enterprise Services Best Practice and Implementation Guide
Editor	Colin Smythe (IMS)
Team Co-Lead	Chris Vento (WebCT Inc.)
Version	1.0
Version Date	11 June 2004
Status	**Final Specification**
Summary	This document presents the IMS Enterprise Services Best Practice and Implementation. The original Enterprise specification was based upon the description of the data model for the information to be exchanged between communicating enterprise systems. The Enterprise Services specification extends this work by adding a series of behavioral models that define how the data models are to be manipulated. The material in this document describes the best practices to be adopted when implementing the Enterprise Services. This version supersedes the IMS Enterprise v1.1 specifications.
Revision Information	11 June 2004
Purpose	This document has been approved by the IMS Technical Board is made available for adoption.
Document Location	http://www.imsglobal.org/es/esv1p0/imses_bestv1p0.html

To register any comments or questions about this specification please visit:
http://www.imsglobal.org/developers/ims/imsforum/categories.cfm?catid=20

List of Contributors

The following individuals contributed to the development of this document:

Name	Organization	Name	Organization
Scott Baker	Oracle Inc.	Les Smith	SCT Inc.
Fred Beshears	UC Berkeley, USA	Colin Smythe	Dunelm Services Ltd.
Kerry Blinco	IMS Australia	Chris Vento	WebCT Inc.
Chris Etesse	Blackboard Inc.	Kimberley Voltero	WebCT Inc.
John Hallet	WebCT Inc.	Scott Wilson	JISC (CETIS), UK
Cathy Schroeder	Microsoft Inc.	Nathaniel Zinn	Blackboard Inc.

Revision History

Version No.	Release Date	Comments
Base Document 1.0	21 July 2003	The final form of the Best Practice and Implementation Guide Base Document. This outline document is submitted to the IMS TB for voting approval and as such is used to show the indicative content of the proposed final document.
Public Draft 1.0	12 January 2004	The final approved Public Draft Document for the IMS Group Management Services Specification.
Final Specification 1.0	11 June 2004	This is the formal Final version of the IMS Enterprise Services Best Practice and Implementation Guide.

Index