ESPRIT Project No. 25 338

Work package E

Personal Profiles

DE3: Personal Profiles Design

ID:	DE3Design	Date:	19/05/98
Author(s):	Steve Battle	Status:	deliverable
Reviewer(s):	INRIA	Distribution:	

Change History

Document Code	Change Description	Author	Date
DE3Design 1.1	Deliverable.	Steve Battle	06/02/98
DE3Design 1.2	Revisions for review	SAB, JPT	19/05/98

1 Introduction

The aim of this document is to combine the user and technical requirements for the management of personal information and through analysis come up with a design that is compatible with the designs of interacting modules. Work-package E (Personal Profiles) relates specifically to the storage and maintenance of so-called *personal* profiles. It has become clear though, that profiles may be necessary to describe other components of the system. It is hoped that many of the techniques developed within this package will be available for re-use within these other areas.

This work-package introduces the idea of content description languages for the specification, storage and communication of information objects. These languages allow us to describe the content of an object in terms of its significant conceptual entities, as distinct from low-level serialisations of particular object instances. These ideas support a standard notation for defining information content; compare with the service interaction work-package which uses similar ideas to form representations of services. Once created, these information objects can be held in an information space for location independent access.

Personal profiles are a means to personal information management. The profiles as defined represent data which is not specific to any application, such as identification or addressing properties. For application-specific information we envisage the use of an extensible data representation which allows the developer to add their own kinds of data.

The following table summarises the timetable for the development of personal profile components.

Profile object

deliverable	***description***
DE5.1	Personal profile object with user interface
DE5.2	Personal diary object with user interface
DE5.3	Finalise generic interface for extensible profiles

2 Requirements and Analysis

2.1 Personal Information

Personal information is all about people and the resources they use. The quality of data appearing in such a profile is generally small and highly irregular. Conventional database techniques, optimised as they are for large amounts of highly regular data, are therefore inappropriate. The **personal profile** would contain the following kinds of information:

- User identification (who am I?)
- Addressing properties (where do I live?)
- communication properties (telephone, fax, email)
- organisational (who do I work for? ; and my job description)
- Security properties (passwords, keys, authorisation)

In addition there is a requirement for an object to handle **diary** information. The many uses to which this could be put include the following:

- The Personal Assistant needs to identify the current location of the user for delivering messages (diary event).
- Agents may need to carry out actions at specified times and intervals (diary action lists and alarms).
- Agents may need to record their actions (diary journal).

However comprehensive the requirements analysis process is at this stage, it is impossible to anticipate all the possible uses and applications to which agents may be put in the future. For this reason we require a representation that is *extensible*, in that new kinds of content may be added once the system is up and running. To show how extensibility works, any additional profile information required by specific pilot applications will be added using these mechanisms. This is covered in the description of so-called **generic profile** data. We need to define a data format that can cope with this wide range of content.

2.2 Domain Object Model

We start with the presentation of the domain object model. Figure 1 lists the main objects that have been identified in relation to personal profiles. A number of non-personal profiles have also been identified in other work-packages. These can benefit from the tools used to develop personal profiles. These include service profiles (service interaction work-package) which include definitions of agent behaviour (missions), and possibly device profiles (user access).

The extensible nature of profile information permits this base set to be expanded at runtime to include additional objects required by particular pilot applications. For example, the stocks agent from pilot application I requires a data structure that contains a description of the user share portfolio, and for pilot application II, a news-agent could assemble a newspaper dynamically based on the interests of the user.

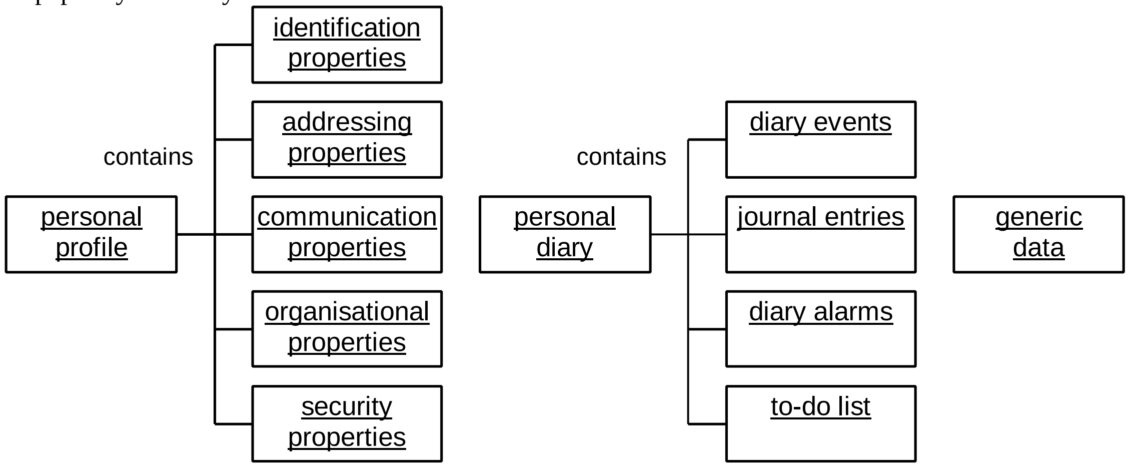

Figure 1 Domain object model for profile data

- **personal profile**: The personal profile is the main repository of personal information.
- **identification properties**: Includes the user's name, date of birth, photograph.
- **addressing properties**: Includes a postal address for services requiring physical delivery.
- **communication properties**: Includes telephone contact numbers and electronic mail addresses.
- **organisational properties**: Includes information about the organisation the user works for, and their role within that organisation.
- **security properties**: Includes public keys.

- **personal diary**: A diary is a co-ordinator of timed activities. The personal diary acts as the user's own calendar, though smaller cut-down versions may be carried by agents to co-ordinate timed activity locally.
- **diary events:** A diary event may be used to specify the whereabouts of the user during a specified period of time, allowing agents to get in touch with them.
- **journal entries**: These entries record events that have happened, and may be used for tracing the activity of agents.
- **diary alarms**: An alarm can be used to activate a given activity at a specific time, and at stated intervals.
- **to-do list**: Unlike an event, an action (to-do) does not extend over a period of time, but must be started and completed within a given interval of time.

- **generic data**: While the personal profile and diary are designed to work across horizontal domains, there is a major requirement for *extensibility*. This means that agent designers and service providers have the means to add their own custom `profile' information once the system is up and running. These may be designed to work in application specific domains. The generic data is in fact a place-holder for this class of objects which cannot be anticipated except in the most general terms.

Use Case Model

Use cases are a way of specifying how a system is actually used. They provide a high level description of system functionality without prescribing a specific class structure. Each *use case* bubble denotes a sequence of actions initiated by an *actor*, which represents a particular role that may be instantiated by a real person or even another system. This document deals only with use cases that have a direct impact profile objects.

Users and agents share similar views of personal information, though there is normally a one-to-one correspondence between users and this particular collection of data. Because this is central to the identity of the user, task agents are normally provided only with read access to personal data. Access to this data may be analysed in terms of a *concrete* set of cases in which the user or agent is able to view and edit specific profile properties. Figure 2 includes a high-level, or *abstract* use-case, 'access personal data', which represents the commonality between these concrete use-cases. Each concrete case extends the functionality of this case.

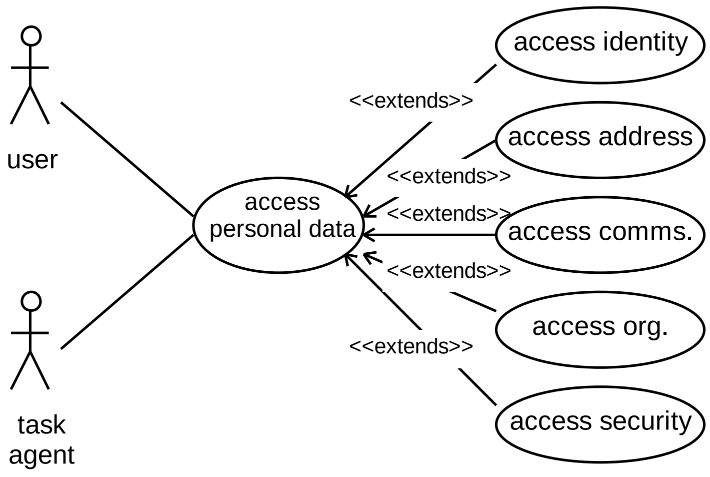

Figure 2 user and agent interaction with the personal profile

- ***access personal data:*** Navigate the interface to the personal data. An abstract use-case representing a point of selection between the various options available.
- ***access identity:*** view/edit user identification details.
- ***access address:*** view/edit user addressing details.
- ***access communications:*** view/edit telecommunication properties.
- ***access organisation:*** view/edit organisational properties.
- ***access security:*** view/edit security properties (e.g. password).

The cases in Figure 3 cover anticipated uses of diary objects from the perspective of both users and agents. The user may use the diary to record their location at specific times so that their personal assistant (see Autonomous Agents work-package) may contact them. They may also set alarms for agents to carry out duties at specific times.

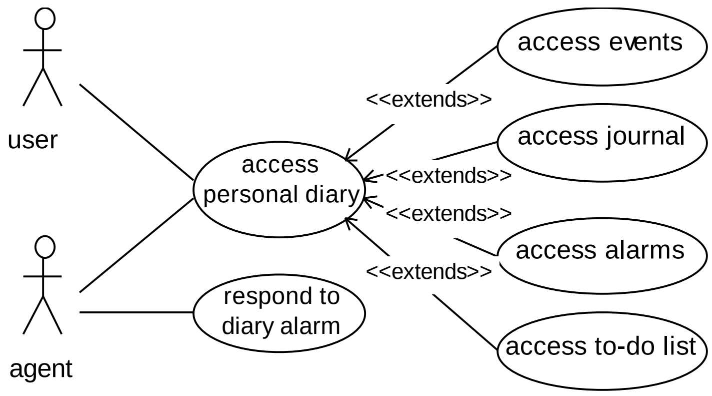

Figure 3 user and agent interaction with the personal diary

- ***access personal diary:*** The user is able to create/edit/delete scheduled diary events and reminders. In addition to being general resource for the user, the diary may be tailored for specific agent behaviour.
- **access events**: View/edit a period of time relating to the user's whereabouts.
- **access journal**: A user may make notes, or an agent may records state information in the diary.
- **access alarms**: View/edit an alarm which may relate to an event or a to-do action item.
- **access to-do list**: View/edit a to-do action-item for either the user or one of their agents.
- **respond to diary alarm**: The diary raises an event which is communicated to any listening agents which respond accordingly.

Figure 4 outlines the requirements for access to generic data, as might be required in pilot applications I and II, and subsequently. The application specific cases could be added to this outline by *extending* 'access generic data'. Examples of how to do this using the scripting language defined in the Autonomous Agents work-package are provided in section 3.6. Before the new data can be used its structure should be defined, so that readers of the new representation can verify that it is well-formed. Both users and agents can access the new data; the agent through a generic interface, and the user through either a generic or custom built graphical user interface.

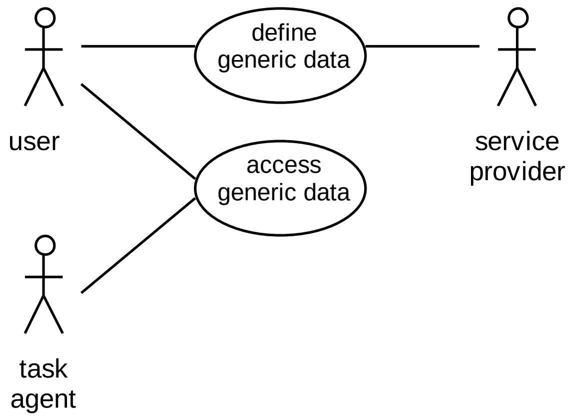

Figure 4 Defining and accessing generic data

- **define generic data**: Define the structure (and possibly implementations) of custom ‘profile’ data.
- **access generic data**: View/edit a custom data structure.

For the objects defined in the domain object model, and which are in some sense common to all applications, we can develop customised interface and control classes. Within this work-package we develop customised interfaces for the personal profile (section 3.1) and personal diary (section 3.3). For generic data objects, we provide the hooks for developers to include their own custom classes, though the primary aim is to build a set of generic accessor functions through which any, and all profile data may be accessed (section 3.5).

3 Design

3.1 Design Methodology

The personal Profiles design methodology is based on Jacobson's three way breakdown of functionality into the separate dimensions of *information* (entity objects), *behaviour* (control objects), and *presentation* (interface objects).

3.1.1 Information

All profile objects are described as structured collections of information objects. Our design addresses the question of how these objects are defined with the idea of a content description language (see section 3.5). As to where, they are stored, because they are not *active* in the sense of being the main focus of a running thread they are ideal candidates for storage in the information space (see the Information Space work-package). These information objects have no function other than to make the information available and to record changes to that information. The information objects defined with tools in the personal profiles work-package will share the same underlying interface.

3.1.2 Behaviour

Any additional functionality required within the personal profile, such as data validation, is described in terms of control objects. In the case of diary objects, this behaviour will include the sourcing of events as specified by alarm data. The information object described using a content description language does not capture any behavioural aspects of the object. To overlay this specific functionality onto generic information objects we may follow one of two courses. The designer may prepare a customised control object written in a standard programming language or they may define scripted control functions as described in section 3.6.

3.1.3 Presentation

While the ordinary control objects are sufficient for access by other objects, there is sometimes a need for direct user interaction with these objects. These interface objects may be defined to work stand-alone, or for on-line access through the personal assistant they must support User Access interfaces. The mapping of an object's content (information object) onto an interface object is essentially a style rule in the sense used by User Access.

An advantage to this three-way break-down of profile functionality is that not all three components need be present in the mobile agent. An agent in the field may carry profile data with it, but may have no need for any presentation components. Even when these presentation objects are needed, it makes sense to locate them close to the user rather than the agent itself. Similarly the agent may not even need to carry large amounts of data with it when all it needs is a control object which handles the connection to a remotely stored information object. thus saving a lot of weight when it comes to agent mobility.

3.2 Personal Profile Design

The content of the profile data is based on the vCard personal data interchange (PDI) specification, which was designed as an electronic business card; containing information about the user and the organisation they work for. A vCard contains properties relating to identification of the user, addressing for postal delivery, telecommunications including email, fax and phone, properties relating to the organisation the user works for, and security. Figure 5 shows the attributes belonging to each class of personal profile object. The figure uses UML (Fowler) class diagram notation (the asterisks refer to the fact that a single profile may contain many (0 or more) addressing and communication objects).

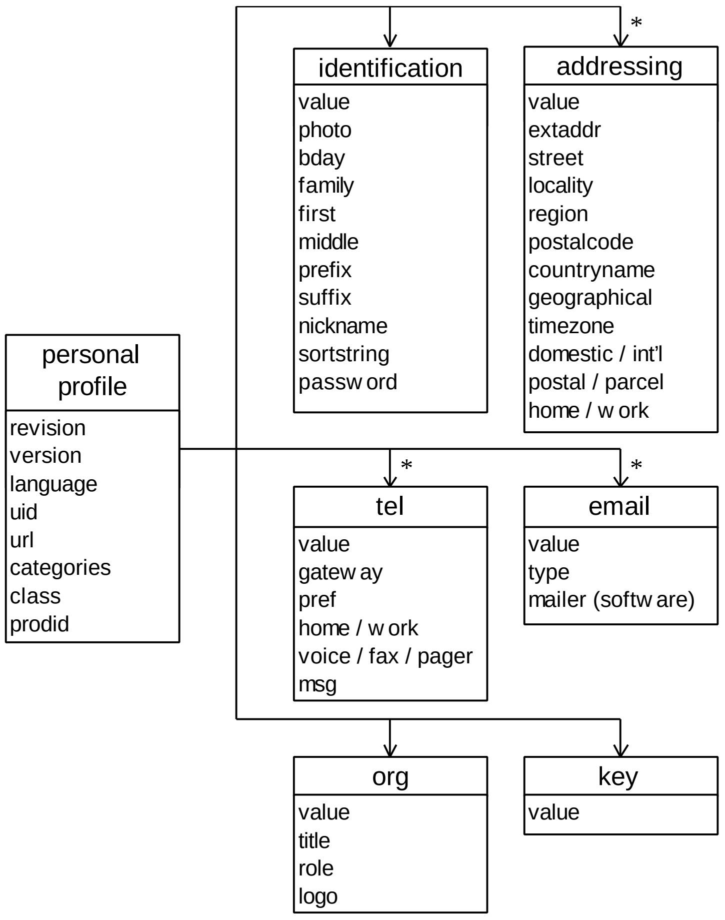

Figure 5 Classification of personal profile information objects

On top of this basic information structure, the control classes provide the main programmatic interface to the personal profile. Where such profile control objects are provided, objects should not communicate directly with the raw profile information objects. The additional behaviour imposed by the profile control enforces simple validation checking on entered fields, and ensures that selected options are mutually consistent. In the personal profile API (Appendix A : Interface Specifications), each attribute is mapped onto a pair of get and set operations. This provides an opportunity for the control classes to perform validation on the assigned values and to respond accordingly. Additionally the API provides methods for accessing sub-elements of the personal profile by class, and where there are many possible sub-elements of the same class they are indexed.

3.2.1 Sequence diagrams

The workings of the personal profile design can be demonstrated with a few simple sequence diagrams. The first diagram in Figure 6 outlines the role of the personal assistant in mediating user access to their personal data. This scenario shows the personal assistant creating a new profile interface and passing it (or a description of it) to User Access. The interface is already filled in with the existing values by getting the relevant attributes from the profile entity, or information object. The interface object is shown as containing a nested profile control object. The control object manages access to the profile entity maintained in an information space.

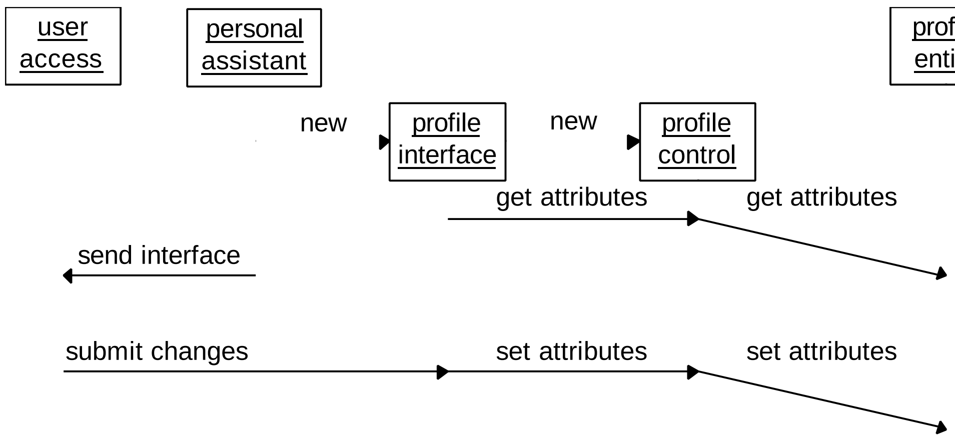

Figure 6 user access to personal data

The second phase of Figure 6 shows the profile data being updated in response to changes made by the user. When the user has filled in a form capturing the user's profile details they click on a submit button which sends an event. This event is interpreted by the interface as a command to set the appropriate attributes on the control object. The communication between the control object and the profile entity is represented diagonally to indicate the potential latency that may occur if the profile entity is held remotely.

An agent's access to the profile data, Figure 7, is considerably simpler, involving only the profile entity itself and a suitable control object. Rather than communicating in a 'batch' mode like the user, the agent may access attributes arbitrarily. Again, latency may occur in the connection between the control object, which is effectively a part of the agent, and the profile entity which may be stored remotely.

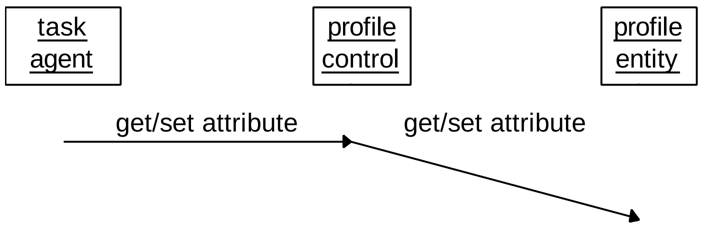

Figure 7 agent access to personal data

3.3 Personal Diary Design

The design of the personal diary is derived from the Internet Calendaring and Scheduling Core Object Specification (iCalendar) which draws heavily on the earlier vCalendar. The vCalendar specification is a set of properties suitable for building diary objects. It is a container for scheduling and reminders. Figure 8 shows the main information objects and their attributes drawn from this specification.

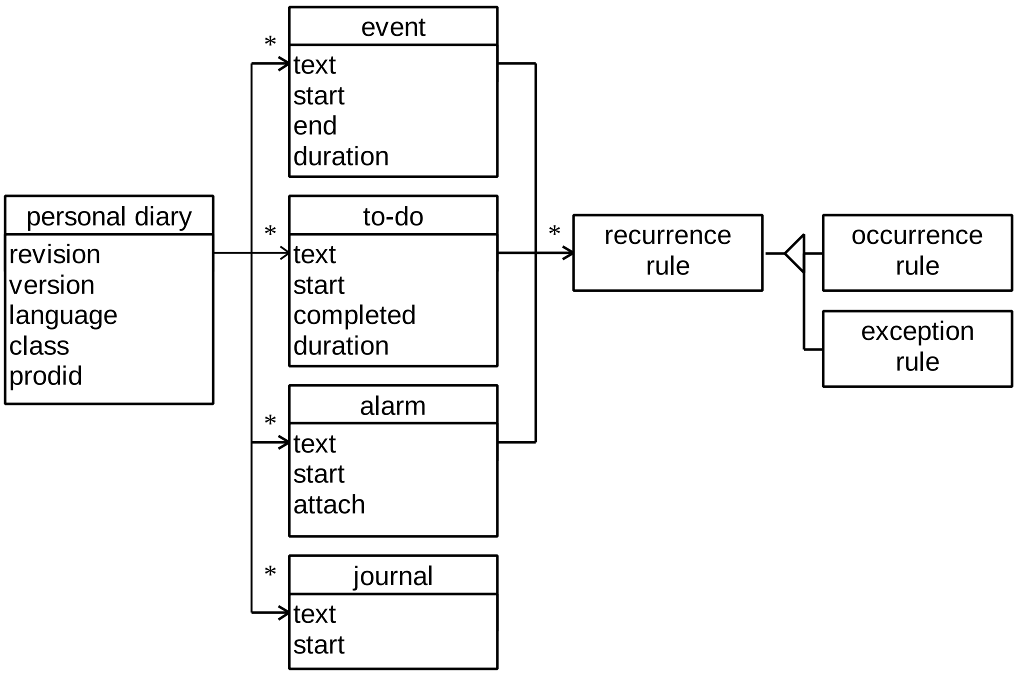

Figure 8 Classification of personal diary information objects

The *start* and *end* attributes are dates and times are represented as ISO 8601 date/time strings, where the month is a two-digit numeric and the time is 24 hour.

<YYYY><MM><DD>T<HH><MM><SS><type>

To avoid problems with local times, Universal time (UTC) is recommended. A UTC time is indicated by appending a 'Z' to the string. For example, the start of the next millenium would be represented by the string "20000101T000000Z".

Events or to-do actions may be associated with a particular *duration*. A duration is represented according to the ISO 8601 standard as the following string. Square brackets indicate optional items. A meeting that was to last for two hours would have a duration represented by, "PT2H".

P[<years>Y][<months>M][<weeks>W][<days>D][T[<hours>H][<minutes>M][<seconds>S]]

The diary also supports ways of representing recurrence rules using XAPIA's CSA specification.

<frequency><interval><frequency modifier>(<end date>|<duration>)

The frequency can be daily (D), weekly (W), monthly (M) or yearly (Y). The interval is a number indicating the number of days/weeks/months/years between recurrences. The modifer depends on the frequency used, and allows the rule to be more specific about the time of recurrence. For example we could specify a daily occurrence by the string "D", but to make this specific to weekdays we could append the modifier "MO,TU,WE,TH,FR". Finally we can specify either an end date or maximum number of occurrences that the rule should repeat for.

- **Alarms**

The diary is the central co-ordinator of timed activities. There may be more than one diary in a system, and in fact, timed event driven activities may be more efficient if a diary that is local to the agent is used. Alarms are set to occur at a specific date and time, and may be set to recur.

- **Journal entries**

A journal entry is a standard way of recording timed information, things that have already happened. As well as recording notes taken by the user, they may also be used to log agent activity.

- **Diary Event**

A diary includes entities called *events* which represent extended periods of time. They might be used to specify the details of a meeting, for example. Their main role within FollowMe is to inform the Personal Assistant of the likely whereabouts of the user. If an urgent message came in from a task agent, the PA might be able to relay the message on to the user by fax, telephone or some other device.

- **To Do reminders**

ToDo reminders support the idea of action-lists which are not tied down to specific times, but might indicate a list of jobs that arise at certain times and need completing within some stated period.

3.3.1 Sequence diagrams

User interaction with the personal diary is much the same as with the personal profile. However the diary control object should perform additional checking to ensure that new diary entries do not overlap with existing entries entries (The word entry is used here to include diary events, journals, and to-do actions). Figure 9 shows just such an interaction with the diary. During the interaction between the diary control and diary entity, it is assumed that other objects are prevented from accessing the object (perhaps by removing it from the information space) so that inconsistencies do not arise.

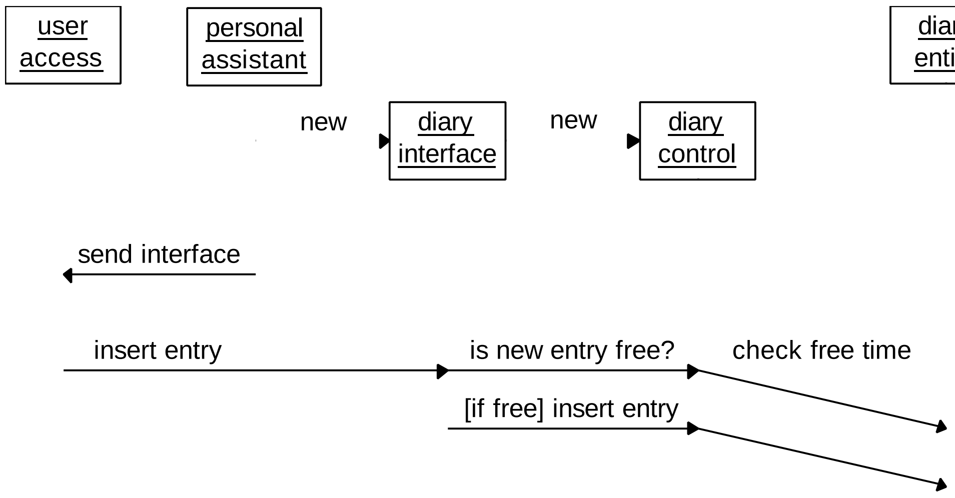

Figure 9 user interaction with diary

Agent interaction with the diary differs significantly from the profile interaction when we consider the behaviours associated with responding to events. A diary containing alarms may be used by agents to wake themselves up and do something useful. Note that events are locally generated within the diary control object which is part of the agent, not from the diary entity itself which exists in the information space. The sequence diagram in Figure 10 is equally applicable to the Personal Assistant as it is to Task Agents.

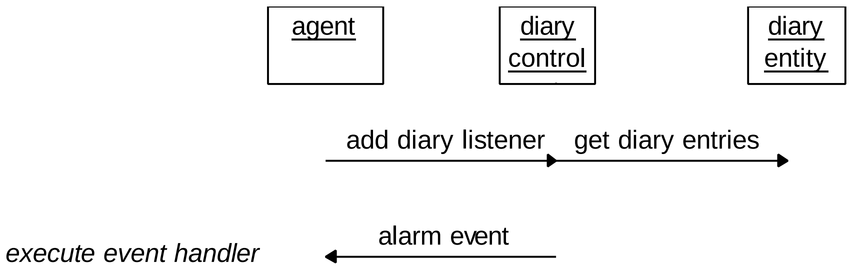

Figure 10 agent response to alarms

An agent responds to an alarm event by executing the appropriate event-handler (see Autonomous Agents work-package).

3.4 Generic profile Design

To design an interface that can be used to access an indeterminate set of information objects, it is necessary to design a conceptual model that will define the way that these information objects are accessed. In the field of databases, the relational model has evolved as a standard way of storing and accessing large numbers of similar data items. For smaller quantities of distinct entities, object-based models are achieving predominance.

However, objects are still difficult to talk about without using specific implementation languages. Interface Definition Languages, such as the CORBA IDL, provide *standard* languages for describing object interfaces. These standard languages provide a way of specifying and documenting object interfaces, and with mappings onto specific implementation languages they benefit inter-operability between heterogenous systems.

Where IDL describes the object interface, we may use what are effectively *content description languages* to talk about the content of an object. The eXtensible Markup Language, XML, is such an example of a content description language and is adopted in this work-package to specify the content of profile objects. *XML is to the content of an object, what IDL is to its interface.*

The XML language (section 3.5.1) is based on the idea of markup, specifically content markup based on the familiar syntax of HTML. Behind the syntax of XML is the model of an object structure as a simple aggregation hierarchy, or tree, which can be explored from the root downwards. Just as IDL has mappings into specific implementation languages, the XML object model is mapped onto specific languages by way of the *Document Object Model*.

3.4.1 Document Object Model

The Document Object Model (DOM) is a development of early programmatic interfaces to documents, found in existing browser technologies. These interfaces are conventionally used by scripts to access HTML elements such as forms and links. This model has been extended for Dynamic HTML (DHTML) to provide full access to the document the script resides in, as an object-based structure. Though the origin of the DOM lies with HTML, its similarities with XML led the DOM working group to adapt the technology to suit XML.

The DOM is an API proposed by a working body hosted by the W3C, which allows programs and scripts to dynamically access and revise the contents of electronic documents. The DOM provides access to the entire contents of an XML document without loss of information. This interface is likely to become an industry standard in the near future, furthering the scope for inter-operability on the web. Early versions of the DOM API underlie prototypes of the personal profile object (deliverable DE5.1). When it becomes sufficiently mature, it will become the standard API for accessing generic XML objects. It includes the basic operations required to allow the user to navigate and edit its contents.

The API is introduced on two levels; higher level interfaces are looked at in section 3.4.2. The first level provides a core interface for HTML and XML documents through a standard programmatic interface. A document defines a tree of objects corresponding to the nesting of markup tags within the document text. The class structure of these objects is illustrated in Figure 11. In addition to methods for accessing data specific to each class of node, each of these inherits from an abstract Node class which provides them with basic functionality to navigate up and down the tree.

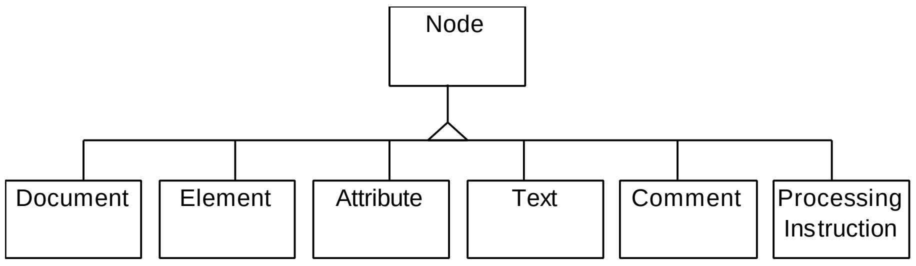

Figure 11 DOM classification

- **Document**

The entire HTML or XML document. Contains a root element.
eg. For an HTML document this is the html tag:- <html>...</html>

- **Element**

A tagged element containing a number of attributes and sub-nodes.
eg. The head element contains a title sub-node:- <head> <title>Document Object Model</title> </head>

- **Attribute**

A named value specified in the opening tag.
eg. A white bgcolor is an attribute of the body element:- <body bgcolor=#ffffff> ... </bgcolor>

- **Text**: Plain text containing no markup.

eg. The text "Document Object Model" of <title>Document Object Model</title>

- **Comment**: A plain text comment.

eg. <!-- Gort, Klaatu Birada Niktoh -->

- **Processing Instruction**

This is a processing instruction intended for the document parser. eg. <?XML version=1.0 ?>. (Processing Instructions are used within FollowMe to define implementation rules for specific tags).

Although the DOM is presented here in the light of HTML and XML, it may also be thought of as a conceptual interface for documents described by other means. For example, it would be possible to implement a DOM interface to standard spreadsheet or word processing files, thus making them accessible to DOM compliant software. There is in fact no reason for the implementation object structure to match this conceptual object structure, indeed for reasons of efficiency this rigid conceptual hierarchy may be undesirable other than for the purposes of communication.

3.4.2 Higher level interfaces

Besides the basic functions available with the standard DOM classes, the generic API will include additional auxiliary classes which simplify the task of navigating around the DOM tree. These may include enumerator and iterator classes.

The DOM API is well suited to conventional programming languages, yet there remains some conceptual mismatch between the DOM and the kinds of interface suitable for scripting. The DOM level one interface promises a mapping from the DOM to scripting models such as ECMAScript. The root Document node provides the functionality of JavaScript's Document object. This includes short-cuts to forms and images which are held in arrays as immediate properties of the Document. A standard interface at this level is essential for the commercial adoption of XML parsers, and will be invaluable in the drive towards a cross-language scripting architecture.

Because HTML is not entirely a thing of the past, the DOM interface may also provide a way for agents to access standard HTML documents and interfaces on the web. Because DOM provides equivalent access to the HTML page as a browser would for the user, agents may access web pages in much the same way. This may prove important for interfacing with legacy services and information sources.

A number of examples utilising iterator interface classes can be found in section 3.6.

3.5 Content Description

A profile is regarded as an expression of the properties of some entity which is independent of any particular object or language implementation. While the same could be said of almost any data file, the use of a standard format for this data opens up new possibilities for data interchange between agents and services. Content description languages let us describe the content of an object in terms of its significant conceptual entities.

Because the content description does not define an object with any significant behaviour, it may be necessary to map the object described onto specific implementation classes (section 3.5.3). The content description is not tied inextricably to a single implementation; it may have alternate interpretations. In the designs considered in this document there are cases where a simple control object is required, and at other times an additional interface object is required for user access.

The content description should not be thought of a low-level representation of an object like an object serialisation, but is a *meta*-description. Like the DOM mapping, there is no strict requirement for the implementation object to preserve the object hierarchy of the content description. There is a clear distinction between implementation objects and the conceptual objects of the content description.

Without content description, the process of constructing an object is often a lengthy affair with an object being built layer by layer, piece by piece. Most languages provide the idea of *literals* for primitive types such as numbers and strings, whereas only list processing languages have traditionally provided the means for stating complex data structures non-procedurally. The declarative representation of objects is therefore a vital accompaniment to the agent scripting language of the Autonomous Agents work-package.

3.5.1 The eXtensible Mark-up Language: XML

The eXtensible Mark-up Language (XML) is a simple language designed by the WorldWide Web Consortium (W3C) for the storage and transmission of information on the internet. XML is a subset of SGML, the Standard Generalised Mark-up Language in which more well known mark-up languages such as HTML are formally defined. However, unlike SGML, XML is small and easy to learn. While HTML has introduced a much needed standardisation across web documents, its function is confined to display mark-up; describing the way things look in a browser window. HTML does not do a good job at representing data; for example, that a given data item is a name rather than an address.

A well formed XML description comprises a hierarchical structure of paired tags, using the <*tag*>...</*tag*> syntax familiar from HTML. The structure is hierarchical in that the tags can be nested to any depth. In addition to this nesting, the opening tag may possess simple attributes. e.g. `<TEL TYPE="FAX">...</TEL>`. While the tags one can use with HTML are fixed by big corporations like Netscape and Microsoft, XML is designed to be extensible. XML does not describe a fixed set of tags, only the structure of well-formed XML. By using a document type declaration (DTD), the XML designer can write rules which determine how their new tags can be arranged. This approach puts power back into the hands of the users, allowing more creative use of tags with customised meanings and functionality.

For example, a particular application may need to extend the core set of tags to include a list of keywords that a particular user is interested in. We could use XML in the following way, listing the user’s interests in science & computing features.

```
<KEYWORDS TYPE=“articles”>
  <LI>science</LI>
  <LI>computing</LI>
</KEYWORDS>
```

Because these new tags have appeared from nowhere, the designer can define how they should be used with a DTD; defining where they may appear, what attributes they may have, and what other tags may be nested within them.

<!ELEMENT KEYWORDS (LI)*> *The KEYWORDS element may contain repeated LI.*
<!ATTLIST KEYWORDS FEATURES CDATA #IMPLIED> *KEYWORDS has an optional FEATURES attribute.*
<!ELEMENT LI (#PCDATA)> *The LI element may contain parsed character data.*

This freedom is balanced by moves to add optional extensions on top of basic XML. These additional layers don't change the language itself but provide standard sets of tags for common uses. Popular extensions include tags for style-sheets and hyper-links which provide XML with HTML like facilities.

3.5.2 Pre-defined Personal Profile and Diary tags

The personal profile has been designed so that each element is available in either a human readable form, or in a *machine readable* form where separate elements are defined. For example, the user name could be specified in any of the three forms below. The first includes the name as normally written, the second differentiates its separate attributes, and the third includes both styles.

```
<PROFILE>
  <ID>Steve Battle</ID>
</PROFILE>

<PROFILE>
  <ID first=“Steve” family=“Battle” />
</PROFILE>

<PROFILE>
  <ID first=“Steve” family=“Battle”>
    Steve Battle
  </ID>
</PROFILE>
```

A similar approach is adopted with the addressing elements where the raw text is useful as a pre-formatted addressing label for the delivery of ordered goods, while the separate address components are more suited to form filling.

```
<PROFILE>
  <ADR
    WORK = “TRUE”
    EXTADDR = “University of the West of England”
    STREET = “Coldharbour Lane”
    LOCALITY = “BRISTOL”
    POSTALCODE = “BS16 1QY”
    COUNTRYNAME = “UNITED KINGDOM”>
    University of the West of England
    Coldharbour Lane
    Bristol, BS16 1QY
  <ADR>
</PROFILE>
```

The diary is the central co-ordinator of timed activities. There may be more than one diary in a system, and in fact, timed event driven activities may be more efficient if a diary that is local to the agent is used. **Alarms** are set to occur at a specific date and time, and may be set to recur. The alarm below sends a scripted birthday card to its user every year (very sad).

```
<DIARY>
  <ALARM START= “19630606” ATTACH= “send(birthdayCard)”/>
    <ORULE FREQ=”Y” />
  </ALARM>
</DIARY>
```

A **journal entry** is a standard way of recording timed information, things that have already happened. As well as recording notes taken by the user, they may also be used to log agent activity. The following example shows a simple journal entry.

```
<DIARY NAME= “Captain’s log”>
  <JOURNAL START= “22591231”>
    Exchanged smalltalk with the Vorlon ambassador today.
  </JOURNAL>
</DIARY>
```

A diary includes entities called ***events*** which represent extended periods of time. They might be used to specify the details of a meeting, for example. Their main role within FollowMe is to inform the Personal Assistant of the likely whereabouts of the user. If an urgent message came in from a task agent, the PA might be able to relay the message on to the user by fax, telephone or some other device.

```
  <DIARY>
    <EVENT START= “19980101T0900” DURATION= “PT8H”>
      Another day older and deeper in debt.
      <ORULE FREQ= “D:MO,TU,WE,TH,FR”/>
    </EVENT>
  </DIARY>
```

ToDo reminders support the idea of action-lists which aren’t tied down to specific times, but might indicate a list of jobs that arise at certain times and need completing within some stated period.

```
<DIARY>
  <TODO>
    <ORULE FREQ=”W” MODIFIER=”MO”>
    Wash-day (blues).
  </TODO>
</DIARY>
```

A full set of Document Type Declarations defining the available tags may be found in appendices D and E.

3.5.3 Implementation rules

The XML specification, shown to the left of Figure 1, is used to derive a ‘profile’ object. Once parsed, that object is regarded as the live copy, at least until the object is closed; saving its contents back into XML. Because XML is fundamentally extensible, we need some way to communicate with arbitrary XML objects. Implementation rules can only be specified in advance. Though it is possible for designers to write their own implementation classes with customised interfaces, it is important to provide ways of accessing user-defined objects without the development of complex Java interfaces. This goal is in line with our aims in the autonomous agents work-package. The aim is to define an interface that is accessible from the task agent language so the script can communicate directly with a generic information object.

Because this is an interface to an information object only, it does no directly support any additional object *behaviour* which represents the semantics of the object. This behaviour is application dependent and so cannot be built into a general-purpose interface.

3.6 Access to Application-specific Information

This section illustrates usage of the Document Object Model (DOM) tool-kit for accessing application-specific information by Task Agents using scripting.

3.7 Example 1 - Simple agent script

The following example is intended to give a flavour of how developers can utilise scripting to access application-specific information represented within a Mission. Consider an application TaskAgent, which requires access to a list of keywords in order to achieve the mission goals. The application-specific information therefore might consist of a single ordered list of textual entries, with attached date. The design process identifies the need to access this information in the following ways:

- `getKeyword(index)` - returns an indexed entry in the keyword list.
- `setKeyword(index, text)` - modifies an existing indexed entry in the keyword list.
- `getDate()` - returns the date associated with the keyword list.

- `setDate(date)` - modifies the date associated with the keyword list.
- `display()` - displays all entries in the keyword list using an existing `writeKeyword` function.

The Mission contains the information as a fragment of XML markup, and this is accessed via utilities provided by the DOM tool-kit. The developer may decide to represent the information as shown in the following markup, using the tags `<KEYWORD>` and `<LI>` coupled with the attribute `DATE` to provide meaning. This is illustrated in the following example fragment:

```
<KEYWORDS DATE="1998-01-20">
  <LI>science</LI>
  <LI>politics</LI>
</KEYWORDS>
```

One scripted implementation of the first function is shown below. This utilises a DOM tool-kit object (Iterator) to provide a linear view onto the XML data. For more detail on the Iterator design pattern, (see Gamma). In this example, two iterators are required - one to access the keyword list (as multiple lists may exist - see example 2 in the next section), and one to access individual list entries. The `toNth` method is used to access an indexed entry in the list of keywords. It should be noted that these code fragments contain no error checking, just basic functionality.

```
function getKeyword(index) {
    keyIterator = new Iterator("KEYWORDS");
    listIterator = new Iterator(keyIterator, "LI");
    return listIterator.toNth(index);
}
```

The next function modifies an existing keyword entry using the `setText` method:

```
function setKeyword(index, text) {
    keyIterator = new Iterator("KEYWORDS");
    listIterator = new Iterator(keyIterator, "LI");
    listIterator.toNth(index).setText(text);
}
```

The date information is held within an attribute of the `<KEYWORD>` tag, and can be accessed in a similar manner:

```
function getDate() {
    keyIterator = new Iterator("KEYWORDS");
    return keyIterator.getAttribute("DATE");
}


function setDate(date) {
    keyIterator = new Iterator("KEYWORDS");
    return keyIterator.setAttribute("DATE", date);
}
```

The DOM toolkit provides an Enumeration object, to enable sequential access of all `<LI>` entries nested within the `<KEYWORDS>` tag. It is assumed that a `writeKeyword` function has been provided for displaying individual entries in a suitable manner.

```
function display() {
    keyIterator = new Iterator("KEYWORDS");
    listEnum = new Enumeration(keyIterator, "LI");
    while (listEnum.hasMoreElements()) {
      writeKeyword(listEnum.nextElement());
    }
}
```

By combining accessor methods such as iterators and enumerators, the developer is free to access and modify any information within the XML document. With suitably scripted methods the designer is free to impose any kind of semantics on their content as they wish, from simple validation to more complex behaviours.

3.8 Example 2 - Additional agent script

This example of scripting extends the scenario described in the previous section. Consider an application TaskAgent, which requires access to categorised lists of keywords in order to achieve the mission goals. The application-specific information therefore might consist of several ordered lists of textual entries, grouped into categories with attached date. The following (example) access methods are required:

- `getKeyword(category, index)` - returns an indexed entry from a particular keyword list.
- `addKeyword(category, keyword)` - creates and adds an entry into a particular keyword list.
- `display()` - displays all entries in all keyword lists using an existing `writeKeyword` method.

The developer may decide to augment the previous markup with an additional `TYPE` attribute to specify the category of keyword list, as shown in this example:

```
<KEYWORDS TYPE="subjects" DATE="1998-01-20">
  <LI>science</LI>
  <LI>politics</LI>
  <LI>lingerie</LI>
</KEYWORDS>

<KEYWORDS TYPE="people" DATE="1998-02-25">
  <LI>Chomsky</LI>
  <LI>Einstein</LI>
  <LI>Crusty the Clown</LI>
</KEYWORDS>
```

The `getKeyword` function utilises the `keyIterator` to obtain the keyword list with matching category:

```
function getKeyword(category, index) {
    keyIterator = new Iterator("KEYWORDS");
    while (keyIterator.getAttribute("TYPE") != category) {
      keyIterator.toNext();
    }
    listIterator = new Iterator(keyIterator, "LI");
    return listIterator.toNth(index);
}
```

Similarly, the `addKeyword` function matches the category prior to adding a new list entry:

```
function addKeyword(category, keyword) {
    keyIterator = new Iterator("KEYWORDS");
    while (keyIterator.getAttribute("TYPE") != category) {
      keyIterator.toNext();
    }
    keyIterator.addElement(new Element("LI", keyword));
}
```

The `display` function uses enumeration to access all list entries in all keyword lists:

```
function display() {
    keyEnum = new Enumeration("KEYWORDS");
    while (keyEnum.hasMoreElements()) {
      keywords = keyEnum.getNext();
      listEnum = new Enumeration(keywords, "LI");
      while (listEnum.hasMoreElement()) {
          writeKeyword(listEnum.getNext());
```

```
        }
     }
}
```

4 Conclusions

The personal profiles work-package demonstrates a flexible approach to the representation of content which conforms to, and stretches emerging industry standards for content description, personal information and diary management, and document models.

The personal profile and diary classes are designed to work across horizontal domains, whereas extensibility offers a way for service providers to create vertical content.

Finally, in contrast to many existing markup viewers, the ability to map between content and implementation classes provides a generic component-based way of interacting with profile data.

5 References

IETF, iCalendar, <http://www.imc.org/draft-ietf-calsch-ical-main>, November 1997.

IETF, vCard MIME Directory Profile, <http://www.imc.org/draft-ietf-asid-mime-vcard >, November 1997.

ChiMu Corporation, MONDO, <http://www.chimu.com/projects/mondo>

Donald G. Firesmith, *Use Cases: the Pros and Cons,* http://www.ksccary.com/usecjrnl.htm

Fowler M, *UML distilled*, Addison-Wesley, 1997 (ISBN:0-201-32563-2).

Gamma E. , *Design Patterns.*

Steven Holzner, XML Complete, McGraw-Hill, 1998.

International Standard Organization, ISO 8601, Representation of dates and times, reference number ISO 8601 : 1988 (E).

Ivar Jacobson, *Object-Oriented Software Engineering: A use-case driven approach*, Addison-Wesley, 1992.

Versit Consortium, vCard: The Electronic Business Card, <http://www.imc.org/pdi>, January 1997

W3C, Document Object Model, <http://www.w3.org/DOM>, December 1997

W3C, Extensible Markup Language (XML), <http://www.w3.org/TR/WD-xml-970807>, August 1997

Appendix A Interface Specifications

Interface UK.ac.uwe.ics.followMe.profile.ProfileInterface

public interface **ProfileInterface**

This defines the interface to a personal profile.
A personal profile provides the means for storing long-term data relating to a FollowMe user. This contains fundamental information that does not change too rapidly.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_adr(int)
Get a structured sequence of address components at a given index.
get_categories()
Get the categories.
get_class()
Get the class.
get_email(int)
Get email address and details.
get_id()
Get the structured components of the name.
get_key(int)
Get the public key.
get_language()
Get the language for profile.
get_org(int)
Set the name of the organization the profile holder works within.
get_prodID()
Get the product ID.
get_revision()
Get the product revision.
get_tel(int)
Get details for telephony communication.
get_uid()
Get the unique id for this profile.
get_url()
Get the URL which is a source of additional information.

get_version()
  Get the version number of the profile specification.
length_adr()
  Get number of Adr objects within profile.
length_email()
  Get number of Email objects within profile.
length_key()
  Get number of Key objects within profile.
length_org()
  Get number of Org objects within profile.
length_tel()
  Get number of Tel objects within profile.
set_adr(AdrInterface, int)
  Set a structured sequence of address components.
set_categories(String)
  Set the categories.
set_class(String)
  Set the class.
set_email(EmailInterface, int)
  Set email address and details.
set_id(IDInterface)
  Set the structured components of the name.
set_key(KeyInterface, int)
  Set the public key.
set_language(String)
  Set the language for profile.
set_org(OrgInterface, int)
  Set the name of the organization the profile holder works within.
set_prodID(String)
  Set the product ID.
set_revision(String)
  Set the product revision.
set_tel(TelInterface, int)
  Set details for telephony communication.
set_uid(String)
  Set the unique id for this profile.
set_url(String)
  Set the URL which is a source of additional information.
set_version(String)
  Set the version number of the profile specification.

Methods

- **set_id**

public abstract void set_id(IDInterface id)

Set the structured components of the name. ID represents the structured components of the name of an individual.

- **get_id**

public abstract IDInterface get_id()

Get the structured components of the name. ID represents the structured components of the name of an individual. This method is normally used in conjunction with set or get methods for ID.

get_id().set_first("Steve") ;

- **set_adr**

public abstract void set_adr(AdrInterface adr,
                             int index)

Set a structured sequence of address components. This is an addressing property. This method would normally be used to set an address to null.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_adr**

public abstract AdrInterface get_adr(int index)

Get a structured sequence of address components at a given index. This is an addressing property. This method is normally used in conjunction with set or get on individual address properties.

get_adr(0).set_locality("Bristol") ;

Parameters:
index - This element may repeat so an index must be supplied.

- **length_adr**

public abstract int length_adr()

Get number of Adr objects within profile.

Returns:
Number of Adr objects.

- **set_tel**

public abstract void set_tel(TelInterface tel, int index)

Set details for telephony communication. This is a telecommunication property.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_tel**

public abstract TelInterface get_tel(int index)

Get details for telephony communication. This is a telecommunication property.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_tel**

public abstract int length_tel()

Get number of Tel objects within profile.

Returns:
Number of Tel objects.

- **set_email**

public abstract void set_email(EmailInterface email, int index)

Set email address and details. This is a telecommunication property.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_email**

public abstract EmailInterface get_email(int index)

Get email address and details. This is a telecommunication property.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_email**

public abstract int length_email()

Get number of Email objects within profile.

Returns:
Number of Email objects.

- **set_org**

public abstract void set_org(OrgInterface org, int, index)

Set the name of the organization the profile holder works within. This is an organizational property.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_org**

public abstract OrgInterface get_org(int index)

Set the name of the organization the profile holder works within. This is an organizational property.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_org**

public abstract int length_org()

Get number of Org objects within profile.

Returns:
Number of Org objects.

- **set_key**

public abstract void set_key(KeyInterface , int index)

Set the public key. This is a security property.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_key**

public abstract KeyInterface get_key(int index)

Get the public key. This is a security property.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_key**

public abstract int length_key()

Get number of Key objects within profile.

Returns:
Number of Key objects.

- **set_revision**

public abstract void set_revision(String rev)

Set the product revision. This records when the profile was last revised. This is an explanatory property.

- **set_version**

public abstract void set_version(String version)

Set the version number of the profile specification. This is an explanatory property.

- **set_language**

public abstract void set_language(String lang)

Set the language for profile. This is an explanatory property.

- **set_uid**

public abstract void set_uid(String uid)

Set the unique id for this profile. This is an explanatory property.

- **set_url**

public abstract void set_url(String url)

Set the URL which is a source of additional information. This is an explanatory property.

- **set_categories**

public abstract void set_categories(String cat)

Set the categories. This is an explanatory property.

- **set_class**

public abstract void set_class(String cl)

Set the class. This is an explanatory property.

- **set_prodID**

public abstract void set_prodID(String prodID)

Set the product ID. This is the identifier of the software that created the profile. This is an explanatory property.

- **get_revision**

public abstract String get_revision()

Get the product revision. This records when the profile was last revised. This is an explanatory property.

- **get_version**

public abstract String get_version()

Get the version number of the profile specification. This is an explanatory property.

- **get_language**

public abstract String get_language()

Get the language for profile. This is an explanatory property.

- **get_uid**

public abstract String get_uid()

Get the unique id for this profile. This is an explanatory property.

- **get_url**

public abstract String get_url()

Get the URL which is a source of additional information. This is an explanatory property.

- **get_categories**

public abstract String get_categories()

Get the categories. This is an explanatory property.

- **get_class**

public abstract String get_class()

Get the class. This is an explanatory property.

- **get_prodID**

public abstract String get_prodID()

Get the product ID. This is the identifier of the software that created the profile. This is an explanatory property.

Interface UK.ac.uwe.ics.followMe.profile.AdrInterface

public interface **AdrInterface**

This defines the interface to an Adr (Address) object.
The Adr represents structured information relating to a single (postal) address.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_countryName()
Get the country within a structured address.
get_dom()
Get the domicile status within a structured address.
get_geo()
Get details of the physical location.
get_home()
Get the home status within a structured address.
get_intl()
Get the international status within a structured address.
get_label()
Get address label.
get_locality()
Get the locality within a structured address.
get_parcel()
Get the parcel status within a structured address.
get_postal()
Get the postal status within a structured address.
get_postalCode()
Get the post code within a structured address.
get_region()
Get the region within a structured address.
get_street()
Get the street name within a structured address.
get_tz()
Get details of the time zone.
get_work()
Get the work status within a structured address.
set_countryName(String)
Set the country within a structured address.
set_dom(boolean)
Set the domicile status within a structured address.

set_geo(String)
  Set details of the physical location.
set_home(boolean)
  Set the home status within a structured address.
set_intl(boolean)
  Set the international status within a structured address.
set_label(String)
  Set address label.
set_locality(String)
  Set the locality within a structured address.
set_parcel(boolean)
  Set the parcel status within a structured address.
set_postal(boolean)
  Set the postal status within a structured address.
set_postalCode(String)
  Set the post code within a structured address.
set_region(String)
  Set the region within a structured address.
set_street(String)
  Set the street name within a structured address.
set_tz(String)
  Set details of the time zone.
set_work(boolean)
  Set the work status within a structured address.

Methods

- **set_street**

public abstract void set_street(String street)

Set the street name within a structured address. This is an addressing property.

- **set_locality**

public abstract void set_locality(String locality)

Set the locality within a structured address. This is an addressing property.

- **set_region**

public abstract void set_region(String region)

Set the region within a structured address. This is an addressing property.

- **set_postalCode**

public abstract void set_postalCode(String postalcode)

Set the post code within a structured address. This is an addressing property.

- **set_countryName**

public abstract void set_countryName(String countryname)

Set the country within a structured address. This is an addressing property.

- **set_dom**

public abstract void set_dom(boolean dom)

Set the domicile status within a structured address. This is an addressing property.

- **set_intl**

public abstract void set_intl(boolean intl)

Set the international status within a structured address. This is an addressing property.

- **set_postal**

public abstract void set_postal(boolean postal)

Set the postal status within a structured address. This is an addressing property.

- **set_parcel**

public abstract void set_parcel(boolean parcel)

Set the parcel status within a structured address. This is an addressing property.

- **set_home**

public abstract void set_home(boolean home)

Set the home status within a structured address. This is an addressing property.

- **set_work**

public abstract void set_work(boolean work)

Set the work status within a structured address. This is an addressing property.

- **set_tz**

public abstract void set_tz(String tz)

Set details of the time zone. This is a geographical property. This is the offset from the prime meridian (GMT).

- **set_geo**

public abstract void set_geo(String geo)

Set details of the physical location. This is a geographical property.

- **set_label**

public abstract void set_label(String label)

Set address label. This is a addressing property.

- **get_tz**

public abstract String get_tz()

Get details of the time zone. This is a geographical property.

- **get_geo**

public abstract String get_geo()

Get details of the physical location. This is a geographical property.

- **get_street**

public abstract String get_street()

Get the street name within a structured address. This is an addressing property.

- **get_locality**

public abstract String get_locality()

Get the locality within a structured address. This is an addressing property.

- **get_region**

public abstract String get_region()

Get the region within a structured address. This is an addressing property.

- **get_postalCode**

public abstract String get_postalCode()

Get the post code within a structured address. This is an addressing property.

- **get_countryName**

public abstract String get_countryName()

Get the country within a structured address. This is an addressing property.

- **get_dom**

public abstract boolean get_dom()

Get the domicile status within a structured address. This is an addressing property.

- **get_intl**

public abstract boolean get_intl()

Get the international status within a structured address. This is an addressing property.

- **get_postal**

public abstract boolean get_postal()

Get the postal status within a structured address. This is an addressing property.

- **get_parcel**

public abstract boolean get_parcel()

Get the parcel status within a structured address. This is an addressing property.

- **get_home**

public abstract boolean get_home()

Get the home status within a structured address. This is an addressing property.

- **get_work**

public abstract boolean get_work()

Get the work status within a structured address. This is an addressing property.

- **get_label**

public abstract String get_label()

Get address label. This is a addressing property.

Interface UK.ac.uwe.ics.followMe.profile.EmailInterface

public interface **EmailInterface**

This defines the interface to an Email (Email address) object.
The Email represents structured information relating to a single electronic-mail address.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_mailer()
Get the type of electronic mail software.
get_text()
Get the electronic mail address.
get_type()
Get the type of electronic mail connection.
set_mailer(String)
Set the type of electronic mail software.
set_text(String)
Set the electronic mail address.
set_type(String)
Set the type of electronic mail connection.

Methods

- **set_text**

public abstract void set_text(String text)

Set the electronic mail address. This is a telecommunications property.

- **set_type**

public abstract void set_type(String type)

Set the type of electronic mail connection. This is a telecommunications property.

- **set_mailer**

public abstract void set_mailer(String mailer)

Set the type of electronic mail software. This is a telecommunications property.

- **get_text**

public abstract String get_text()

Get the electronic mail address. This is a telecommunications property.

- **get_type**

public abstract String get_type()

Get the type of electronic mail connection. This is a telecommunications property.

- **get_mailer**

public abstract String get_mailer()

Get the type of electronic mail software. This is a telecommunications property.

Interface UK.ac.uwe.ics.followMe.profile.IDInterface

public interface **IDInterface**

This defines the interface to a ID (user identifier).
An ID provides the means for storing details identifying a FollowMe user.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_bday()
Get the birthdate of the individual.
get_family()
Get the family name within a structured name.
get_first()

Get the first name within a structured name.
get_fn()
Get the formatted name.
get_middle()
Get the middle name within a structured name.
get_nickname()
Get the informal name.
get_password()
Get the password required for access to the personal assistant.
get_photo()
Get a reference to an image of the individual.
get_prefix()
Get the name prefix within a structured name.
get_sortstring()
Get the sort string.
get_suffix()
Get the name suffix within a structured name.
set_bday(String)
Set the birthdate of the individual.
set_family(String)
Set the family name within a structured name.
set_first(String)
Set the first name within a structured name.
set_fn(String)
Set the formatted text corresponding to the name.
set_middle(String)
Set the middle name within a structured name.
set_nickname(String)
Set an informal name.
set_password(String)
Get the password required for access to the personal assistant.
set_photo(String)
Set a reference to a graphics file; an image of the individual This is an identification property.
set_prefix(String)
Set the name prefix within a structured name.
set_sortstring(String)
Set the sort string.
set_suffix(String)
Set the name suffix within a structured name.

Methods

- **set_fn**

public abstract void set_fn(String fn)

Set the formatted text corresponding to the name. This is an identification property.

- **set_nickname**

public abstract void set_nickname(String nickname)

Set an informal name. This is an identification property.

- **set_photo**

public abstract void set_photo(String photo)

Set a reference to a graphics file; an image of the individual This is an identification property.

- **set_bday**

public abstract void set_bday(String bday)

Set the birthdate of the individual. This is an identification property. e.g. set_bday("1963-06-06T06:00:00Z")

- **set_family**

public abstract void set_family(String family)

Set the family name within a structured name. This is an identification property.

- **set_first**

public abstract void set_first(String first)

Set the first name within a structured name. This is an identification property.

- **set_middle**

public abstract void set_middle(String middle)

Set the middle name within a structured name. This is an identification property.

- **set_prefix**

public abstract void set_prefix(String prefix)

Set the name prefix within a structured name. This is an identification property.

- **set_suffix**

public abstract void set_suffix(String suffix)

Set the name suffix within a structured name. This is an identification property.

- **set_password**

public abstract void set_password(String passwd)

Get the password required for access to the personal assistant. This is a security property.

- **set_sortstring**

public abstract void set_sortstring(String sortstring)

Set the sort string. This is an alternative name for use by sorting programs. This is an explanatory property.

- **get_fn**

public abstract String get_fn()

Get the formatted name. This is an identification property.

- **get_nickname**

public abstract String get_nickname()

Get the informal name. This is an identification property.

- **get_photo**

public abstract String get_photo()

Get a reference to an image of the individual. This is an identification property.

- **get_bday**

public abstract String get_bday()

Get the birthdate of the individual. This is an identification property.

- **get_family**

public abstract String get_family()

Get the family name within a structured name. This is an identification property.

- **get_first**

public abstract String get_first()

Get the first name within a structured name. This is an identification property.

- **get_middle**

public abstract String get_middle()

Get the middle name within a structured name. This is an identification property.

- **get_prefix**

public abstract String get_prefix()

Get the name prefix within a structured name. This is an identification property.

- **get_suffix**

public abstract String get_suffix()

Get the name suffix within a structured name. This is an identification property.

- **get_password**

public abstract String get_password()

Get the password required for access to the personal assistant. This is a security property.

- **get_sortstring**

public abstract String get_sortstring()

Get the sort string. This is an alternative name for use by sorting programs. This is an explanatory property.

Interface UK.ac.uwe.ics.followMe.profile.KeyInterface

public interface **KeyInterface**

This defines the interface to an Key (public key) object.
The Key represents structured information relating to a single public encryption key.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_text()
Get the public key string.
get_type()
Get the public key type.
set_text(String)
Set the public key string.
set_type(String)
Set the public key type.

Methods

- **set_text**

public abstract void set_text(String text)

Set the public key string. This is a security property.

- **set_type**

public abstract void set_type(String type)

Set the public key type. This is a security property.

- **get_text**

public abstract String get_text()

Get the public key string. This is a security property.

- **get_type**

public abstract String get_type()

Get the public key type. This is a security property.

Interface UK.ac.uwe.ics.followMe.profile.OrgInterface

public interface **OrgInterface**

This defines the interface to a Org (organisation details).
An Org provides the means for storing details relating to an organisation.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_logo()
Get a reference to a logo associated with the organisation.
get_orgname()
Get the name of the organisation the profile holder works within.
get_role()
Set the role or occupation within the organisation.
get_title()
Set the job title.
set_logo(String)
Get a reference to a logo associated with the organisation.
set_orgname(String)
Set the name of the organisation the profile holder works within.
set_role(String)
Set the role or occupation within the organisation.
set_title(String)
Set the job title.

Methods

- **set_orgname**

public abstract void set_orgname(String name)

Set the name of the organisation the profile holder works within. This is an organisational property.

- **set_title**

public abstract void set_title(String title)

Set the job title. This is an organisational property.

- **set_role**

public abstract void set_role(String role)

Set the role or occupation within the organisation. This is an organisational property.

- **set_logo**

public abstract void set_logo(String logo)

Get a reference to a logo associated with the organisation. This is an organisational property.

- **get_orgname**

public abstract String get_orgname()

Get the name of the organisation the profile holder works within. This is an organisational property.

- **get_title**

public abstract String get_title()

Set the job title. This is an organisational property.

- **get_role**

public abstract String get_role()

Set the role or occupation within the organisation. This is an organisational property.

- **get_logo**

public abstract String get_logo()

Get a reference to a logo associated with the organisation. This is an organisational property.

Interface UK.ac.uwe.ics.followMe.profile.TelInterface

public interface **TelInterface**

This defines the interface to an Tel (telecommunications details) object.
The Tel represents structured information for a single telecommunication access details.
FollowMe WorkPackage E - Personal Profiles

Method Index

get_cell()
Get the cell status.
get_fax()
Get the fax status.
get_gateway()
Get the telephone gateway.
get_home()
Get the home status.
get_msg()
Get the msg status.
get_pager()
Get the pager status.
get_pref()
Get the preferred status.
get_text()
Get the telephone number property.

get_voice()
Get the voice status.
get_work()
Get the work status.
set_cell(boolean)
Set the cell status.
set_fax(boolean)
Set the fax status.
set_gateway(String)
Set the telephone gateway.
set_home(boolean)
Set the home status.
set_msg(boolean)
Set the msg status.
set_pager(boolean)
Set the pager status.
set_pref(boolean)
Set the preferred status.
set_text(String)
Set the telephone number property.
set_voice(boolean)
Set the voice status.
set_work(boolean)
Set the preferred status.

Methods

- **set_text**

public abstract void set_text(String text)

Set the telephone number property. This is a telecommunications property.

- **set_gateway**

public abstract void set_gateway(String gateway)

Set the telephone gateway. This is a telecommunications property.

- **set_pref**

public abstract void set_pref(boolean pref)

Set the preferred status. This is a telecommunications property.

- **set_work**

public abstract void set_work(boolean work)

Set the preferred status. This is a telecommunications property.

- **set_home**

public abstract void set_home(boolean home)

Set the home status. This is a telecommunications property.

- **set_voice**

public abstract void set_voice(boolean voice)

Set the voice status. This is a telecommunications property.

- **set_fax**

public abstract void set_fax(boolean fax)

Set the fax status. This is a telecommunications property.

- **set_msg**

public abstract void set_msg(boolean msg)

Set the msg status. This is a telecommunications property.

- **set_cell**

public abstract void set_cell(boolean cell)

Set the cell status. This is a telecommunications property.

- **set_pager**

public abstract void set_pager(boolean pager)

Set the pager status. This is a telecommunications property.

- **get_text**

public abstract String get_text()

Get the telephone number property. This is a telecommunications property.

- **get_gateway**

public abstract String get_gateway()

Get the telephone gateway. This is a telecommunications property.

- **get_pref**

public abstract boolean get_pref()

Get the preferred status. This is a telecommunications property.

- **get_work**

public abstract boolean get_work()

Get the work status. This is a telecommunications property.

- **get_home**

public abstract boolean get_home()

Get the home status. This is a telecommunications property.

- **get_voice**

public abstract boolean get_voice()

Get the voice status. This is a telecommunications property.

- **get_fax**

public abstract boolean get_fax()

Get the fax status. This is a telecommunications property.

- **get_msg**

public abstract boolean get_msg()

Get the msg status. This is a telecommunications property.

- **get_cell**

public abstract boolean get_cell()

Get the cell status. This is a telecommunications property.

- **get_pager**

public abstract boolean get_pager()

Get the pager status. This is a telecommunications property.

Appendix B Personal Diary

Interface UK.ac.uwe.ics.followMe.diary.DiaryInterface

public interface **DiaryInterface**

This defines the interface to a Diary object.
FollowMe WorkPackage E - Personal Profiles

Method Index

addDiaryListener(DiaryListener)
  Add a listener to this diary
get_alarm(int)
  Get an alarm from the diary.
get_event(int)
  Get an event from the diary.
get_journal(int)
  Get a journal from the diary.
get_language()
  Get the language for diary.
get_prodID()
  Get the product ID.
get_revision()
  Get the product revision.
get_toDo(int)
  Get an ToDo from the diary.
get_version()
  Get the version number of the diary specification.
length_alarm()
  Get number of alarm within diary.
length_event()
  Get number of events within diary.
length_journal()
  Get number of journal entries in the diary.
length_toDo()
  Get number of ToDo within diary.
removeDiaryListener(DiaryListener)

Remove a listener from this diary
set_alarm(AlarmInterface, int)
Set an alarm within the diary.
set_event(EventInterface, int)
Set an event within the diary.
set_journal(JournalInterface, int)
Set a journal within the diary.
set_language(String)
Set the language for diary.
set_prodID(String)
Set the product ID.
set_revision(String)
Set the product revision.
set_toDo(ToDoInterface, int)
Set an ToDo within the diary.
set_version(String)
Set the version number of the diary specification.

Methods

- **addDiaryListener**

public abstract void addDiaryListener(DiaryListener l)

Add a listener to this diary

Parameters:
l - Listener to add.

- **removeDiaryListener**

public abstract void removeDiaryListener(DiaryListener l)

Remove a listener from this diary

Parameters:
l - Listener to remove.

- **set_journal**

```
public abstract void set_journal(JournalInterface j,
                                 int index)
```

Set a journal within the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_journal**

public abstract JournalInterface get_journal(int index)

Get a journal from the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_journal**

public abstract int length_journal()

Get number of journal entries in the diary.

Returns:
Number of journals.

- **set_alarm**

```
public abstract void set_alarm(AlarmInterface a,
                               int index)
```

Set an alarm within the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_alarm**

public abstract AlarmInterface get_alarm(int index)

Get an alarm from the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_alarm**

public abstract int length_alarm()

Get number of alarm within diary.

Returns:
Number of alarms.

- **set_event**

```
public abstract void set_event(EventInterface e,
                               int index)
```

Set an event within the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_event**

public abstract EventInterface get_event(int index)

Get an event from the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_event**

public abstract int length_event()

Get number of events within diary.

Returns:
Number of events.

- **set_toDo**

```
public abstract void set_toDo(ToDoInterface t,
                              int index)
```

Set an ToDo within the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **get_toDo**

```
public abstract ToDoInterface get_toDo(int index)
```

Get an ToDo from the diary.

Parameters:
index - This element may repeat so an index must be supplied.

- **length_toDo**

```
public abstract int length_toDo()
```

Get number of ToDo within diary.

Returns:
Number of ToDo objects.

- **set_revision**

```
public abstract void set_revision(String rev)
```

Set the product revision. This records when the diary was last revised. This is an explanatory property.

- **get_revision**

```
public abstract String get_revision()
```

Get the product revision. This records when the diary was last revised. This is an explanatory property.

- **set_version**

```
public abstract void set_version(String version)
```

Set the version number of the diary specification. This is an explanatory property.

- **get_version**

```
public abstract String get_version()
```

Get the version number of the diary specification. This is an explanatory property.

- **set_language**

public abstract void set_language(String lang)

Set the language for diary. This is an explanatory property.

- **get_language**

public abstract String get_language()

Get the language for diary. This is an explanatory property.

- **set_prodID**

public abstract void set_prodID(String prodID)

Set the product ID. This is the identifier of the software that created the diary. This is an explanatory property.

- **get_prodID**

public abstract String get_prodID()

Get the product ID. This is the identifier of the software that created the diary. This is an explanatory property.

Interface UK.ac.uwe.ics.followMe.diary.AlarmInterface

public interface **AlarmInterface** extends JournalInterface

This defines the interface to an Alarm object.
This class represents a (possibly repeating) instantaneous occurrence, that raises a DiaryEvent (cf. alarm call)
FollowMe WorkPackage E - Personal Profiles

Method Index

get_attach()
Get the message attached to the alarm event.
get_exception(int)
Get exception rule.
get_occurrence(int)
Get occurrence rule.
length_exception()
Get number of exception rules within this alarm.
length_occurrence()
Get number of occurrence rules within this alarm.
set_attach(String)
Set the message attached to the alarm event.
set_exception(RecurrenceRule, int)
Set exception rule.
set_occurrence(RecurrenceRule, int)
Set occurrence rule.

Methods

- **set_attach**

public abstract void set_attach(String message)

Set the message attached to the alarm event.

- **get_attach**

public abstract String get_attach()

Get the message attached to the alarm event.

- **set_occurrence**

```
public abstract void set_occurrence(RecurrenceRule rule,
                                    int index)
```

Set occurrence rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **get_occurrence**

public abstract RecurrenceRule get_occurrence(int index)

Get occurrence rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **length_occurrence**

public abstract int length_occurrence()

Get number of occurrence rules within this alarm.

Returns:
Number of occurrence rules.

- **set_exception**

```
public abstract void set_exception(RecurrenceRule rule,
                                   int index)
```

Set exception rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **get_exception**

public abstract RecurrenceRule get_exception(int index)

Get exception rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **length_exception**

public abstract int length_exception()

Get number of exception rules within this alarm.

Returns:
Number of exception rules.

Interface UK.ac.uwe.ics.followMe.diary.JournalInterface

public interface **JournalInterface**

This defines the interface to an Journal () object.
This class represents a single. instantaneous, timestamped occurrence (cf journal entry).
FollowMe WorkPackage E - Personal Profiles

Method Index

get_start()
Get the start DateTime.
get_text()
Get the textual entry.
set_start(DateTime)
Set the start DateTime.
set_text(String)
Set the textual entry.

Methods

- **set_text**

public abstract void set_text(String text)

Set the textual entry.

- **get_text**

public abstract String get_text()

Get the textual entry.

- **set_start**

public abstract void set_start(DateTime dt)

Set the start DateTime.

- **get_start**

public abstract DateTime get_start()

Get the start DateTime.

Interface UK.ac.uwe.ics.followMe.diary.EventInterface

public interface **EventInterface** extends JournalInterface

This defines the interface to an Event object.
This represents an occurence with finite duration, hence contains seperate start & finish times (cf a business meeting).
FollowMe WorkPackage E - Personal Profiles

Method Index

get_alarm(int)
Get an alarm.
get_duration()
Get the event duration.
get_end()
Get the end DateTime.
get_exception(int)
Get exception rule.
get_occurrence(int)
Get occurrence rule.
length_alarm()
Get number of alarms attached to this event.
length_exception()
Get number of exception rules within this alarm.
length_occurrence()
Get number of occurrence rules within this alarm.
set_alarm(AlarmInterface, int)
Set an alarm.
set_duration(DateTime)
Set the event duration.
set_end(DateTime)
Set the end DateTime.
set_exception(RecurrenceRule, int)
Set exception rule.
set_occurrence(RecurrenceRule, int)
Set occurrence rule.

Methods

- **set_end**

public abstract void set_end(DateTime dt)

Set the end DateTime.

- **get_end**

public abstract DateTime get_end()

Get the end DateTime.

- **set_duration**

public abstract void set_duration(DateTime dt)

Set the event duration.

- **get_duration**

public abstract DateTime get_duration()

Get the event duration.

- **set_alarm**

```
public abstract void set_alarm(AlarmInterface alarm,
                               int index)
```

Set an alarm.

Parameters:
index - Multiple alarms may exist so an index must be supplied.

- **get_alarm**

public abstract AlarmInterface get_alarm(int index)

Get an alarm.

Parameters:
index - Multiple alarms may exist so an index must be supplied.

- **length_alarm**

public abstract int length_alarm()

Get number of alarms attached to this event.

Returns:
Number of alarms.

- **set_occurrence**

```
public abstract void set_occurrence(RecurrenceRule rule,
                                    int index)
```

Set occurrence rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **get_occurrence**

public abstract RecurrenceRule get_occurrence(int index)

Get occurrence rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **length_occurrence**

public abstract int length_occurrence()

Get number of occurrence rules within this alarm.

Returns:
Number of occurrence rules.

- **set_exception**

```
public abstract void set_exception(RecurrenceRule rule,
                                   int index)
```

Set exception rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **get_exception**

public abstract RecurrenceRule get_exception(int index)

Get exception rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **length_exception**

public abstract int length_exception()

Get number of exception rules within this alarm.

Returns:
Number of exception rules.

Interface UK.ac.uwe.ics.followMe.diary.ToDoInterface

public interface **ToDoInterface** extends JournalInterface

This defines the interface to a ToDo object.
This class represents an action to be performed (cf the completion of a report).
FollowMe WorkPackage E - Personal Profiles

Method Index

get_alarm(int)
  Get an alarm.
get_completed()
  Get the end DateTime.
get_duration()
  Get the event duration.
get_exception(int)
  Get exception rule.
get_occurrence(int)
  Get occurrence rule.
length_alarm()
  Get number of alarms attached to this event.
length_exception()
  Get number of exception rules within this alarm.
length_occurrence()
  Get number of occurrence rules within this alarm.
set_alarm(AlarmInterface, int)
  Set an alarm.
set_completed(DateTime)
  Set the completed DateTime.
set_duration(DateTime)
  Set the event duration.
set_exception(RecurrenceRule, int)
  Set exception rule.
set_occurrence(RecurrenceRule, int)
  Set occurrence rule.

Methods

- **set_completed**

public abstract void set_completed(DateTime dt)

  Set the completed DateTime.

- **get_completed**

public abstract DateTime get_completed()

  Get the end DateTime.

- **set_duration**

public abstract void set_duration(DateTime dt)

Set the event duration.

- **get_duration**

public abstract DateTime get_duration()

Get the event duration.

- **set_alarm**

```
public abstract void set_alarm(AlarmInterface alarm,
                               int index)
```

Set an alarm.

Parameters:
index - Multiple alarms may exist so an index must be supplied.

- **get_alarm**

public abstract AlarmInterface get_alarm(int index)

Get an alarm.

Parameters:
index - Multiple alarms may exist so an index must be supplied.

- **length_alarm**

public abstract int length_alarm()

Get number of alarms attached to this event.

Returns:
Number of alarms.

- **set_occurrence**

```
public abstract void set_occurrence(RecurrenceRule rule,
                                    int index)
```

Set occurrence rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **get_occurrence**

public abstract RecurrenceRule get_occurrence(int index)

Get occurrence rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **length_occurrence**

public abstract int length_occurrence()

Get number of occurrence rules within this alarm.

Returns:
Number of occurrence rules.

- **set_exception**

```
public abstract void set_exception(RecurrenceRule rule,
                                   int index)
```

Set exception rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **get_exception**

```
public abstract RecurrenceRule get_exception(int index)
```

Get exception rule.

Parameters:
index - Multiple rules may exist so an index must be supplied.

- **length_exception**

```
public abstract int length_exception()
```

Get number of exception rules within this alarm.

Returns:
Number of exception rules.

Appendix C Element definitions

The following element definitions define the structure of a well formed profile described in XML.

Each type of element has an element declaration. Following the element name we may list (in brackets) the valid sub-elements. Each sub-element may be followed by a character that specifies if it can appear one or more times (+), zero or more times (*), or zero or one times(?). The ordering of these elements is not specified.

<!ELEMENT element_name (sub-element1, ... , sub-elementn) >

If the element start and end tags can enclose text, this is regarded as a special sub-element called #PCDATA. An arbitrary mix of text and tagged sub-elements is described as follows.

<!ELEMENT element_name (#PCDATA | sub-element1 | ... | sub-elementn)* >

If the element has no valid sub-elements (including textual elements), the list may be replaced by EMPTY.

<!ELEMENT element_name EMPTY >

Each element may also possess a number of attributes which are listed separately. The attribute types used here specify either character data (CDATA) or an enumerated sequence of values, within brackets separated by '|'. The default section indicates either a default value, or indicates whether the attribute is mandatory (#REQUIRED), or may be omitted and left up to the reader to determine the appropriate value (#IMPLIED).

<!ATTLIST element_name attribute_name type default >

The following stategy has been followed in converting from the vCard and iCalendar specifications into XML. Structured data types are represented by distinct sub-elements. An example of this is the structured name, N. Non-structured non-repeating data types are represented as attributes of their parent element. An example of this is the formatted name, FN, shown here as an attribute of PROFILE. Where non-structured properties are repeatable an element with a simple value attribute is introduced. An example of this is the Agent element that simply specifies an agent identifier, though any number of agents may be specified. Where vCard specifies notes, comments,or descriptions, this has been mapped onto the #PCDATA sub-element. Where such free-form text is allowed, an optional LANGUAGE attribute can be used.

Appendix D profile DTD

```
<!-- FollowMe Profile DTD -->
<!-- jpt modified 8/5/98 -->
<!DOCTYPE PROFILE [

<!ELEMENT PROFILE (ID,(ADR|TEL|EMAIL|ORG|KEY)*)>
        <!ATTLIST PROFILE
                REV CDATA #IMPLIED
                VERSION CDATA #IMPLIED
                LANGUAGE CDATA #IMPLIED
                UID CDATA #IMPLIED
                URL CDATA #IMPLIED
                CATEGORIES CDATA #IMPLIED
                CLASS (PUBLIC|PRIVATE) "PUBLIC"
                PRODID CDATA #IMPLIED>
        <!ELEMENT ID (#PCDATA)*>
        <!ATTLIST ID
                PHOTO CDATA #IMPLIED
                BDAY CDATA #IMPLIED
                FAMILY CDATA #IMPLIED
                FIRST CDATA #IMPLIED
                MIDDLE CDATA #IMPLIED
                PREFIX CDATA #IMPLIED
                SUFFIX CDATA #IMPLIED
                NICKNAME CDATA #IMPLIED
                SORTSTRING CDATA #IMPLIED
                PASSWORD CDATA #IMPLIED>
        <!ELEMENT ADR (#PCDATA)*>
        <!ATTLIST ADR
                EXTADDR CDATA #IMPLIED
                STREET CDATA #IMPLIED
                LOCALITY CDATA #IMPLIED
                REGION CDATA #IMPLIED
                POSTALCODE CDATA #IMPLIED
                COUNTRYNAME CDATA #IMPLIED
                GEO CDATA #IMPLIED
                TZ CDATA #IMPLIED
                DOM (TRUE|FALSE) "FALSE"
                INTL (TRUE|FALSE) "FALSE"
                POSTAL (TRUE|FALSE) "FALSE"
                PARCEL (TRUE|FALSE) "FALSE"
                HOME (TRUE|FALSE) "FALSE"
                WORK (TRUE|FALSE) "FALSE">
        <!ELEMENT TEL (#PCDATA)*>
        <!ATTLIST TEL
                GATEWAY CDATA #IMPLIED
                PREF (TRUE|FALSE) "TRUE"
                WORK (TRUE|FALSE) "FALSE"
                HOME (TRUE|FALSE) "FALSE"
                VOICE (TRUE|FALSE) "FALSE"
                FAX (TRUE|FALSE) "FALSE"
                MSG (TRUE|FALSE) "FALSE"
                CELL (TRUE|FALSE) "FALSE"
                PAGER (TRUE|FALSE) "FALSE">
        <!ELEMENT EMAIL (#PCDATA)*>
        <!ATTLIST EMAIL
                TYPE CDATA #IMPLIED
                MAILER CDATA #IMPLIED>
        <!ELEMENT ORG (#PCDATA)*>
        <!ATTLIST ORG
                TITLE CDATA #IMPLIED
                ROLE CDATA #IMPLIED
                LOGO CDATA #IMPLIED>
        <!ELEMENT KEY (#PCDATA)*>
        <!ATTLIST KEY
                TYPE CDATA #IMPLIED>
]>
```

Appendix E Diary DTD

```
<!-- FollowMe Diary DTD -->
<!-- jpt modified 8/5/98 -->
<!DOCTYPE DIARY [

<!ELEMENT DIARY       (JOURNAL | ALARM | EVENT | TODO)*>

  <!ATTLIST DIARY     REVISION      CDATA #IMPLIED>
  <!ATTLIST DIARY     VERSION               CDATA #IMPLIED>
  <!ATTLIST DIARY     LANGUAGE      CDATA #IMPLIED>
  <!ATTLIST DIARY     CLASS         (PUBLIC|PRIVATE) "PUBLIC"
  <!ATTLIST DIARY     PRODID                CDATA #IMPLIED>


<!-- JOURNAL - single, instantaneous, timestamped occurrence -->
<!ELEMENT JOURNAL (#PCDATA)>

  <!ATTLIST JOURNAL  START          CDATA #REQUIRED>


<!-- ALARM - possibly repeating instantaneous occurrence, that raises a DiaryEvent -->
<!ELEMENT ALARM (#PCDATA | ORULE | EXRULE)* >

  <!ATTLIST ALARM     START         CDATA #REQUIRED>
  <!ATTLIST ALARM     ATTACH                CDATA #IMPLIED>


<!-- EVENT - occurence with finite duration (eg. start & finish or duration) -->
<!ELEMENT EVENT       (#PCDATA | ORULE | EXRULE | ALARM)* >

  <!ATTLIST EVENT     START         CDATA #REQUIRED>
  <!ATTLIST EVENT     END           CDATA #IMPLIED>
  <!ATTLIST EVENT     DURATION      CDATA #IMPLIED>


<!-- TODO - action to be performed -->
<!ELEMENT TODO        (#PCDATA | ORULE | EXRULE | ALARM)* >

  <!ATTLIST TODO      START         CDATA #REQUIRED>
  <!ATTLIST TODO      COMPLETED     CDATA #IMPLIED>
  <!ATTLIST TODO      DURATION      CDATA #IMPLIED>
<!-- ORULE - Ocurrence rule -->
<!ELEMENT ORULE EMPTY >

  <!ATTLIST ORULE      VALUE          CDATA #REQUIRED>


<!-- EXRULE - Exception rule -->
<!ELEMENT EXRULE EMPTY >

  <!ATTLIST EXRULE     VALUE          CDATA #REQUIRED>


]>
```

Appendix F Example XML profile

This XML profile expresses that same information as the profile in the appendix of document DE1Survey. The mapping to XML is more concise in the current version.

```
<PROFILE REV = “19980515T144500Z” URL = “www.ics.uwe.ac.uk/~sab” >

  <ID
    FAMILY = “Battle”
    FIRST = “Steven”
    MIDDLE = “Andrew”
    PREFIX = “Dr.”
    BDAY = “19630606” >
    Steve Battle
  </ID>

  <ADR
    TZ = “0000”
    GEO = “51.47,-2.58” >
  </ADR>

  <ORG
    TITLE = “Researcher”
    LOGO = “http://zen.btc.uwe.ac.uk/icons/hilogo2.gif” >
    The University of the West of England,
    The Intelligent Computer Systems Centre
  </ORG>

  <ADR
    WORK = TRUE
    EXTADDR = “UWE”
    STREET = “Coldharbour Lane, Frenchay”
    LOCALITY = “Bristol”
    REGION = “South Gloucestershire”
    POSTALCODE = “BS16 1QY”
    COUNTRYNAME = “United Kingdom” />

  <TEL WORK = “TRUE”>+44-117-965-6261x3177”</TEL>
  <TEL FAX = “TRUE”>+44-117-975-0416”</TEL>

  <EMAIL>sab@ics.uwe.ac.uk</EMAIL>

</PROFILE>
```