13/15 rue des Ateliers 1080 Bruxelles, Belgium
Tel.+32-75-235-984, fax.+32-2-414-8404
http://www.imatix.com
info@imatix.com

Technical White Paper

iAF Object Access Language

The iMatix Application Framework (iAF) is a design and construction process for three-tier web-based or GUI applications. The iAF Repository stores the design model of an application and drives a template-based code generation and documentation process. The iAF Repository has three layers: presentation, business objects, and database. These layers are described using XML framework languages: the Presentation Framework Language, the Object Framework Language, and the Database Framework Language.

Communications between the presentation and business-logic layers are conducted using a XML protocol language, called the **Object Access Language (OAL)**. OAL defines the messages passed between front-end and back-end. This document formally specifies the Object Access Language syntax.

Copyright

Version Information

Written:	15 January 2000
Revised:	23 January 2000

Disclaimer

The Object Access Language (OAL)

General Operating Principles

- OAL defines a *communications protocol* between a *front-end application* that handles presentation and a *back-end application* that handles business objects.
- OAL is designed to work with applications built using the iMatix Application Framework. Specifically, it is designed to access business objects defined using the Object Framework Language (OFL). The designs of OFL and OAL are intertwined. However: OFL is mainly used by the application developer, while OAL is mainly used by the technical designer when constructing code-generation templates.
- OAL is an *XML language* that is capable of expressing complex data structures.
- OAL describes a set of *messages* that are passed between the front-end and back-end applications. Each OAL message has a specific purpose.
- OAL messages are exchanged in a *fully synchronous* manner – each exchange consists of one request sent from the front-end to the back-end, and one reply from the back-end to the front-end.
- OAL *does not specify a transport mechanism* – the front-end and back-end applications can communicate using remote procedure calls, direct subroutine calls, asynchronous message queues, etc. All these transport mechanisms can be used to carry OAL messages.
- An OAL message is sent to a specific object handler. OAL does not specify how this handler is identified – we assume that naming conventions or lookup mechanisms will allow the front-end application to identify the correct handler for each object.
- OAL messages *are not validated* using formal XML validation techniques (DTDs). Any party receiving and decoding an OAL message should ignore any items or attributes that it does not understand.
- OAL assumes that the back-end server is *context-free*. That is, an OAL request message is never dependant on some previous step in the front-end to back-end session.

XML Syntax Notes

- XML is case-sensitive.
- White spaces are not significant around '=', but are significant in values. Item names and attribute names must be delimited by spaces, =, >, or <.
- A valid XML tree has exactly one item at the root level.

OAL Request Messages

An OAL request from front-end to back-end is an XML string of this format:

```
<oal do = "request" attributes…>
    [<child items>…]
</oal>
```

The attributes and child items in each message depend on the specific request. The possible requests are:

create	Create a new object instance
fetch	Fetch an object instance
update	Update an object instance
delete	Delete an object instance
state	Get information about the object's workflow state
search	Fetch a list of objects according to some search criteria
execute	Apply an object method

An invalid request results in a return string containing:

```
<oal done="error" cause="oa">
```

The Create Request

```
<oal do     = "create"
     user   = "userid"
   [ id     = "newid" ]
     object data
    />
```

The create request creates a new object instance. The **id** attribute specifies the identifier for the new object in cases when an id is not assigned automatically. An object may not already exist with this identifier. The **user** attribute is required so that the object handler can check authority and maintain audit information.

The back-end server responds to a create request with one of these messages:

```
<oal done = "ok"
     id   = "objectid" />
```

The ok reply returns an id for the created object.

```
<oal done     = "error"
     cause    = "db"|"ua"|"id"
     message = "error message" />
```

The error reply indicates that the object could not be created. The cause indicates the reason for the problem: "db" means that there is a database problem of some kind. "ua" means that the user does not have sufficient authority to create a new object. "id" means that the provided id (for an object where ids are not assigned) was invalid or already existed. A full error message is provided, for logging or display if needed.

The Fetch Request

```
<oal do   = "fetch"
     id   = "objectid"
     user = "userid"
     view = "viewname" />
```

The fetch request retrieves a particular object instance. Object instances are identified by the **id** attribute. In most cases this will be a numeric value. In some cases it will be a short text value. The **user** attribute holds the current user id. The **view** name specifies which view to fetch.

The back-end server responds to a fetch request with one of these messages:

```
<oal done     = “ok”
     id       = “objectid”
     revised  = “timestamp”
     access   = “r”|”rw”|”rwd”
     view     = “viewname” >
     object data
</oal>
```

The ok reply returns data for the requested object. The **revised** attribute provides the timestamp for the object, and must be returned to the object handler with any ‘update’ request. The **access attribute may** be one of: r (read-only), rw (read-write), or rwd (read-write-delete) and tells the front-end whether the user can modify and/or delete the object data or not. The object data consists of a tree of child items, each bearing the name of a data table, and attributes matching the fields in each data table.

```
<oal done    = “error”
     cause   = “nf”|”nv”|“db”|”ua”|”wf”
     message = “error message” />
```

The error reply indicates that the object could not be accessed. The cause indicates the reason for the problem: “nf” means the object does not exist. “nv” means that the view does not exist. “db” means that there is a database problem of some kind. “ua” means that the user does not have sufficient authority to read the object. “wf” means that the request was not allowed by the object’s workflow. A full error message is provided, for logging or display if needed.

The Update Request

```
<oal do       = “update”
     id       = “objectid”
     revised  = “timestamp”
     user     = “userid”
     view     = “viewname”
     object data
</oal>
```

The update request updates a particular object instance. The object must have been fetched by a fetch request or a search request. The **revised** attribute contains the timestamp of last revision, as provided by the last fetch message. The **view** attribute specifies which view to update.

The back-end server responds to an update request with one of these messages:

```
<oal done = “ok” />
```

The ok reply indicates that the update was successful.

```
<oal done    = “error”
     cause   = “nf”|”nv”|“db”|”ua”|”wf”|”ro”|”ac”
     message = “error message” />
```

The error reply indicates that the object could not be updated. The cause indicates the reason for the problem: “nf” means the object does not exist. “nv” means that the view does not exist. “db” means that there is a database problem of some kind. “ua” means that the user does not have sufficient authority to change the object. “wf” means that the request was not allowed by the object’s workflow. “ro” means that the object is read-only. “ac” means that the object was updated by another user in the meantime, and that this user’s changes cannot be accepted. A full error message is provided, for logging or display if needed.

The Delete Request

```
<oal do   = "delete"
     id   = "objectid"
     user = "userid" />
```

The delete request deletes an object instance. The **id** attribute specifies the identifier for the object. The **user** attribute is required so that the object handler can check authority and maintain audit information.

The back-end server responds to a create request with one of these messages:

```
<oal done = "ok"/>
```

```
<oal done    = "error"
     cause   = "db"|"ua"|"id"|"wf"
     message = "error message" />
```

The error reply indicates that the object could not be created. The cause indicates the reason for the problem: “db” means that there is a database problem of some kind. “ua” means that the user does not have sufficient authority to delete the object. “id” means that the provided id (for an object where ids are not assigned) was invalid. “wf” means that the request was not allowed by the object’s workflow. A full error message is provided, for logging or display if needed.

The State Request

```
<oal do = “state”
     id = “objectid” />
```

The state request requests information about the object’s workflow state.

The back-end server responds to an update request with one of these messages:

```
<oal done = "ok"
     state = "statename" >
    <method name = "methodname" />…
</oal>
```

The ok reply provides the object's current state name, and the list of methods that are allowed on the object in its current state. This information can be used to control actions shown to the user. The state information should never be used for programmatic purposes, but purely as cosmetic information: the names of workflow states can be changed arbitrarily by the object designer.

```
<oal done    = "error"
     cause   = "nf"
     message = "error message" />
```

The error reply indicates that the object does not exist. The cause attribute is always "nf". A full error message is provided, for logging or display if needed.

The Search Request

```
<oal do      = "search"
   [ search  = "searchname" ]
   [ control = "first"|"last"|"next"|"previous" ]
   [ base    = "baseid" ]
   [ limit   = "number" ]
   [ user    = "userid" ] >
   [ <criterionname>criterionvalue</criterionname>
    … ]
</oal>
```

The search request asks the object handler to provide a list of objects matching some search criteria. The default search is according to the object's primary index (this search is usually called "summary"). The **control** attribute tells the object handler how to control the search. "first" and "last" return the start and end of the object list. "next" and "previous" return the next or previous part of the list starting from some 'base' value. The **base** attribute specifies this value. The base is always outside the search list – for instance, a base of '0' implies that the first object is '1' or higher. The **limit** attribute tells the object handler how many objects to return. By default this is 20.

The **user** attribute is optionally used to check authority. Objects may be protected and searchable only by certain user levels and types. In this case the front-end must provide the user id.

The criteria list is a list of arbitrary item names, used to provide information for the search. These criteria are established by prior agreement between the object handler and the presentation layer.

The back-end server responds to an update request with one of these messages:

```
<oal done    = "ok"
     count   = "searchsize"
     view    = "viewname"
     access  = "r"|"rw"|"rwd"
     <object id = "objectid" revised = "timestamp" >
        object data
```

```
          </object>…
</oal>
```

The ok reply returns data for the requested objects. The **updates** attribute tells the front-end whether the search view is updateable or not (whether the user can modify the object data or not). By default this is "false". The view name used for the search is also provided. The object data consists of a list of objects, each containing a tree of child items, bearing the name of a data table, and attributes matching the fields in each data table.

Each object has an **id** attribute and a **revised** attribute – these must be returned to the object handler with any 'update' requests.

```
<oal  done    = "error"
      cause   = "ns"|"db"|"ua"|"tm"
      message = "error message" />
```

The error reply indicates that the search could not be done. The cause indicates the reason for the problem: "nv" means that the search does not exist. "db" means that there is a database problem of some kind. "ua" means that the user does not have sufficient authority to do the search. "tm" means that the search hit a time-out limit. A full error message is provided, for logging or display if needed.

The Execute Request

```
<oal  do     = "execute"
      id     = "objectid"
      method = "methodname"
      user   = "userid" >
    [ <criterionname>criterionvalue</criterionname>
     … ]
</oal>
```

The method request asks the object handler to execute a workflow method on the object. The method must be one of the methods accepted by the object in its current state.

The back-end server responds to an execute request with one of these messages:

```
<oal  done = "ok" />
```

The ok reply indicates that the object handler executed the method correctly.

```
<oal  done    = "error"
      cause   = "nf"|"db"|"ua"|"wf"
      message = "error message" />
```

The error reply indicates that the method could not be executed. The cause indicates the reason for the problem: "nf" means that the object does not exist. "db" means that there is a database problem of some kind. "ua" means that the user does not have sufficient authority to execute the method. "wf" means that the method was not allowed by the object's workflow. A full error message is provided, for logging or display if needed.