iMatix Corporation sprl

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 Framework 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.

This document formally specifies the Object Framework Language (OFL) syntax.

Copyright

Version Information

Written: 15 November 1999
Revised: 29 January 2000

Disclaimer

The information contained in this document is distributed on an “as-is” basis without any warranty either expressed or implied. The customer is responsible for the use of this information and/or implementation of any techniques mentioned. iMatix Corporation has reviewed the information for accuracy, but there is no guarantee that a customer using the these techniques and/or information will obtain the same or similar results in its own operating environment.

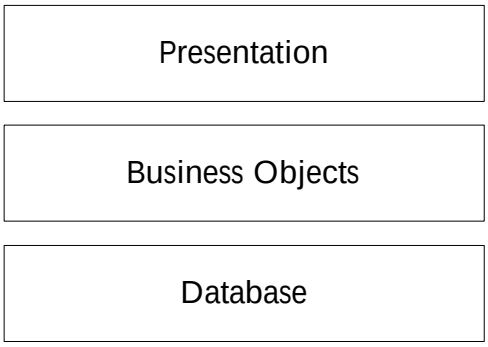
It is possible that this material may contain references to, or information about, iMatix Corporation products or services that have not been announced. Such references or information must not be construed to mean that iMatix intends to announce such products or services.

iMatix Corporation retains the title to the copyright in this paper, as well as title to the copyright in all underlying works. iMatix Corporation retains the right to make derivative works and to republish and distribute this paper to whoever it chooses to.

The iAF Repository Architecture

Repository Layers

The iAF Repository uses three separate layers of definition to describe applications:



The Database Definition Layer

This describes the physical database, and everything that can be usefully attached to this. We describe the tables, table columns, and domains (data types) for each column. We define the indexes on tables, and relationships between tables. In many cases, the database definition can be extracted automatically from an existing database (called 'reverse engineering').

We use the Database Framework Language (DFL) to describe the database definition layer.

The Object Definition Layer

This describes the business objects (we use objects only in this sense). An object consists of a set of database tables, and some intelligence. The object data can be shown in one of multiple 'views', possibly depending on its state.

We use the Object Framework Language (OFL) to describe the object definition layer.

The Presentation Definition Layer

This describes the user-interface: what data is shown, what actions are possible, and how the user-interface is constructed from screens.

We use the Presentation Framework Language (PFL) to describe the presentation definition layer.

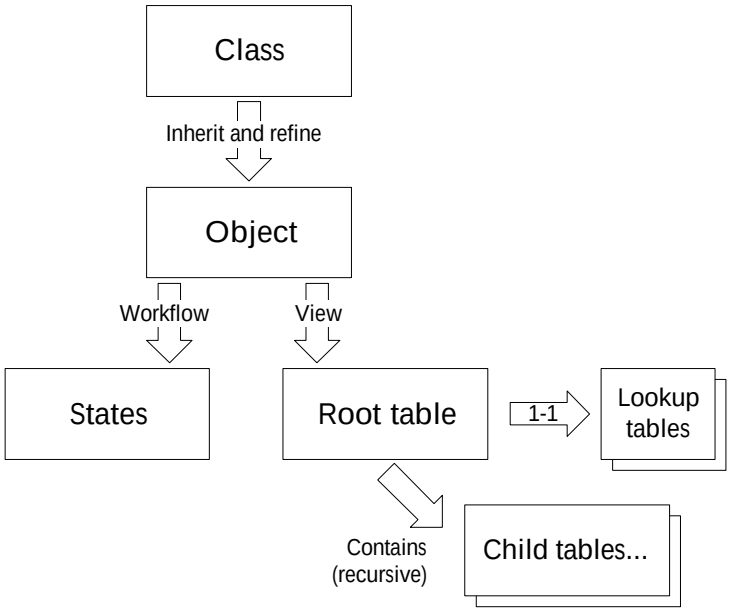
The Object Framework Language (OFL)

Terminology

- An *application* is a set of related objects.
- An *object* is a set of tables, accessed from one *root table*, and a set of *methods* that implement some business logic.
- Each object is handled by a program or component called an *object handler*.
- An object can be accessed through a number of *views*. Each view defines a set of tables and data that is passed to and from the object handler.
- An object can exist in a number of different *states*. These states define what views are available, and what methods are applicable, at any moment.
- Object *classes* define common patterns for objects that can be used as the basis for the states and views used in an object.

Object Composition

The OFL defines objects in terms of classes, states, and tables as shown in this diagram:



An object centralises all handling for one database table or a related set of database tables. This can range from the very simple ("this object implements a simple update model on this single table") to the very complex ("this object implements a customised workflow model on this set of tables").

We define each object in terms of:

1. The tables that compose the object.
2. The views that let the client layer access the object's data.
3. The searches that let the client layer define sets of objects for listing, scrolling, etc.
4. The states that define the object's workflow.

Object Tables

An object always refers to one 'root' table. The identifier of an object instance is equivalent to the identifier of a record in the root table. Tables that refer to objects – such as those storing workflow information – refer to the identifier of the root table record.

The root table can contain child tables. A table is considered to be a child table if it has a 'childof' or 'multiplex' link in the DFL. Child tables are attached to the root table in a '1-to-N' relationship. The root table can also contain reference tables. A reference table translates some code, e.g. country code. Reference tables are attached to the root table in a '1-to-1' relationship.

Each child table can itself contain child tables and reference tables, ad infinitum. An object can thus be arbitrarily complex.

By default, if no tables are specified for an object, we assume that the object maps directly to the database table with the same name.

Object Views

The object is accessed through one or more views. Each view can work with different database tables, always starting with the object root table. Views can be read-write (the default), or read-only.

Object Searches

An object search provides a mechanism for listing and scrolling through objects. Each search specifies an index and selection criteria. Searches are based on views, which specify what data to return for each object found.

Object States

Any object is always in a well-defined 'state'. The state determines what actions are allowed (or expected) on the object at any moment. All objects have at least one state. In the simplest case, an object is always in the same state. In more complex cases, objects can move from state to state depending on the actions that the user executes. We call such complex cases 'workflow objects'. To handle workflow objects, the application will use special tables to keep track of each object's current state and history of states.

Overall Structure of OFL Files

OFL files have the extension ".ofl". This is their complete potential XML structure:

```
ofl
    object
        view
            table …
                field
        search
            criterion
        state
            method
                action
    class
        search
            criterion
        state
            view
                method
            method
                action
    include
    inherit
    …rule
```

One OFL file defines one **application** item and zero or more **object** items. The DFL file can also contain other items beneath the application item, as this document describes.

Rules

Rules provide non-core instructions about the way the objects are used in application programs. Applications that use the OFL may use or ignore rule information as appropriate. The full set of rules is formalised here so that there is common agreement about the way that rules are specified and (if used) implemented.

Rules can be attached to most items. A rule item has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the rule. The meaning of each rule name depends on the context it is used in – i.e. the entity it is attached to, and the rule condition used, if any.
when	(optional)	Specifies the rule condition.
what	(optional)	Specifies an option argument name.
value	(optional)	Specifies an optional argument value, used when the rule defines some optional criteria.

The Ofl Item

This item defines the application. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the application. This name may be used for comments, documentation.
description	name	A one-line description.
written	mandatory	The date that the OFL was written, in YYYYMMDD format.
revised	mandatory	The date that the OFL was revised, in YYYYMMDD format.
author	mandatory	The person responsible for the OFL, which we recommend to be of the form “First-name Surname <email-address>”.
script	(optional)	The default code generation script to use. If you do not specify this in the OFL file, you can specify it externally when running GSLgen.
dfl	(optional)	The name of the application DFL file. An application is always based on a single principal DFL file. This can itself be composed of multiple DFL files though the use of the DFL inherit and include functions.

The ofl item value (the text between <ofl> and </ofl>) is a longer description of the application, if this is required. This description can be used when generating documentation. It is assumed to be plain-text, without mark-up tags.

The Object Item

This item defines an application business object. In general you will define one object per database table, except where tables are grouped together in a parent-to-child relationship.

The object item has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the table, as created in the application. It must be unique within the application definition.
description	name	A one-line description.
handler	(optional)	Name of the object handler, which is a subroutine, component, or service. The object handler name should be a valid short filename without file extension. If omitted, this comes from the object root table name.
item	(optional)	A short descriptive name for the object. If this is not specified, it comes from the root table **description** attribute in the DFL file.

The object item value is a longer description of the object, if this is required. This description can be used when generating documentation. It is assumed to be plain-text, without mark-up tags.

The Object View Item

This item defines an object view. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the view.
access	rwd	Can be one of: r (read-only), rw (read-write), or rwd (read-write-delete). When the view is being shown, the access defines some of the actions that the end user can apply to it. Other actions come from the workflow methods that are available in the object's current state.

Each object always has at least three views: "create", used to create new objects, "summary", used for searches on the object, and "detail" used for normal access to the object. These views, and a "summary" search, are created automatically as required for all objects based on the default class.

The View Table Item

This item defines an object view table. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the view.

The View Table Field Item

This item defines a field in an object view table. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the view.

The Object Search Item

This item defines an object search. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the search.
view	search name	
index	primary	Name of index used in searches.
inverse	0	Do we use an inverse search order? By default the search is done in ascending index order.

Each object always has at least one search called “summary”. This is used for searches on the object. If this search is not specified, it is assumed.

The Search Criterion Item

This item defines an object search criterion. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the search criterion.
default	(optional)	Default value for the criterion.

The Object State Item

This item defines an object state. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the state.
inherit	true	If ‘true’, the state inherits its structure from the object class state with the same name. This means that any methods

		and views present in the class state are included in the object state, though they can be overridden by definitions in the object state.

The State Method Item

This item defines a method, valid in a specific object state. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the method.
nextstate	Current state	

The State Method Action Item

This item defines an action executed when a method is requested in the object state. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the action.

The State View Item

This item defines a view available in a specific object state. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the method.
access	The object view access attribute	Can be one of: r (read-only), rw (read-write), or rwd (read-write-delete).

The Class Item

This item defines an object class. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the class.
default	false	if “true”, this class is used as the default class for all objects. If two or more classes are defined as default classes, the first in order is used. Classes included or inherited from other OFL files come after classes defined in the OFL itself.

The class item can contain states and searches. Objects based on this class inherit the states and searches specified for the class.

The Inherit Item

This item defines a OFL file that is used as a basis for the current OFL. It has these attributes:

Attribute:	Default:	Has this purpose:
filename	(mandatory)	Specifies the name of the OFL file to inherit. The file must either be in the current directory, or be specified with a full path name.

An inherited OFL file is a full and valid OFL, complete with <ofl> item. All classes in this file are copied into the current OFL, if they are not already present. The definitions of tables in the inherited OFL are not used. If the inherited OFL contains <include> or <inherit> tags, these are ignored. Note that <inherit> can only be defined as a child of the root <ofl> item.

The Include Item

This item defines a OFL subfile that is included into the current OFL. It has these attributes:

Attribute:	Default:	Has this purpose:
filename	(mandatory)	Specifies the name of the file to include. The file must either be in the current directory, or be specified with a full path name.

An included OFL file must conform to certain rules. Firstly, it does not contain a 'ofl' item, and is thus a non-conforming XML file (it may have multiple items at the root level). We call this a 'partial' XML file. Secondly, all definitions in the included file must be 'safe' with respect to the current OFL. For example, two objects with the same name are not permitted – this causes an error.

The <include> tag is identical to typing the contents of the included file at that location. Note that <include> can only be defined as a child of the root <ofl> item.

Worked Example

...