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 Database 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 Database Framework Language (DFL) syntax.

Copyright

Copyright © 1999-2000 iMatix Corporation. This document may not be distributed, copied, archived, printed, photocopied, or transmitted in any way whatsoever without prior permission from iMatix Corporation. All rights are reserved.

IMATIX® is a registered trademark of iMatix Corporation. All other trademarks are the property of their respective owners.

Version Information

Written: 15 November 1999
Revised: 30 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.

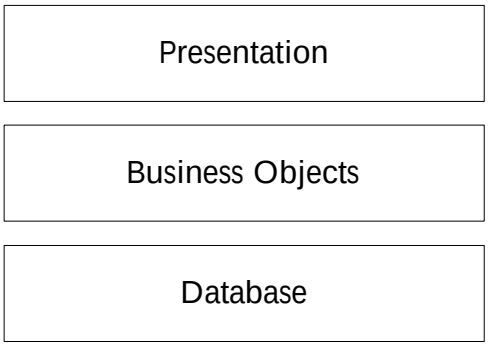
It is possible that this material may contain references to, or information about, iMatix Corporation products or services that have not been announced. Such references or information must not be construed to mean that iMatix intends to announce such products or services.

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 Database Framework Language (DFL)

Terminology

- A *database* is a set of related data tables that are stored together, accessed through some common security model, etc. One database may be equivalent to all or part of a real relational database, a workspace, etc. depending on the database product used.
- A *database table* is one physical database table, composed from a number of *fields*, and a number of *indexes*. All database tables have at least one unique index, called the *primary index*. In most cases, this is a unique numeric identifier, but it can also be a 'natural key' such as the ISO code for a country.
- Indexes can be composed from several fields – this is especially useful for constructing *alternate indexes* that let the application access the table data in some order other than the primary index order.
- Database tables have relationships, called *links* to other tables. There are several types of link, covering the types of relationship used in databases: parent-to-child links, and references to other tables.
- Field may be based on a *domain*, which specific the physical data type for the field, and possibly a set of enumerated *values*. Domains may be grouped out of several fields.

Overall Structure of DFL Files

DFL files have the extension ".dfl". This is their complete potential XML structure:

```
dfl
    table
        field
        index
            field
        link
            field
    domain
        value
        link
            field
    include
    inherit
    …rule
```

One DFL file defines one dfl item and zero or more table items. The DFL file can also contain other items beneath the dfl 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. The names of the rule conditions, and what they mean, depends on each rule.
what	(optional)	Specifies an option argument name.
value	(optional)	Specifies an optional argument value, used when the rule defines some optional criteria.

Rules provide non-core instructions about the way the database is used in application programs. These instructions do not define the database structure, as such, but are conveniently attached to domains, tables, field, links, and other items that compose the DFL.

Applications that use the DFL 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.

Clean Identifiers

Clean identifiers are a technique that allow databases to be defined in a simpler and more abstracted fashion. When we use clean identifiers, the primary index for most or all tables is a single numeric identifier, assigned incrementally each time a record is created. Such identifiers are never reused, and are large enough that they never roll-over. This technique allows references to records to consist just of their identifiers. It simplifies many aspects of data access, and is especially useful when we use a code generation approach.

When we reverse-engineer an existing database, this notion may not apply, so the DFL allows clean identifiers to be switched-off on a table by table basis.

The DFL makes use of clean identifiers to assume certain things about the names used for index fields:

- The primary index for tables is a single field called ‘id’ unless another primary index is specified.

- All reference links (when one table refers to another) are based on the primary index field ('id' or whatever other field is defined as the primary key), unless specified otherwise.
- All parent-to-child links are based on the id field (in the parent table) and a field called parentid in the child table. The combination (parentid, id) is defined automatically as an alternate index in the child table.

You can omit index specifications for tables that use clean identifiers. To use clean identifiers, specify a 'clean id' rule at the dfl level, e.g.:

```
<dfl name = "demo">
    <rule name = "clean ids" />
```

The Dfl Item

This item defines the database. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the database. This name may be used for comments, documentation.
description	name	A one-line description.
written	mandatory	The date that the DFL was written, in YYYYMMDD format.
revised	mandatory	The date that the DFL was revised, in YYYYMMDD format.
author	mandatory	The person responsible for the DFL, 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 DFL file, you can specify it externally when running GSLgen.
realname	name	The real, physical name of the database. The default value for this attribute is the database name.
prefix	(optional)	If specified, the real names of all database tables are prefixed by these letters. You should use the prefix item in general to ensure that multiple applications that share a database do not have conflicts over the names of tables.
userid	(optional)	The user id used to access the database during development.
password	(optional)	The password used to access the database during development.

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

The user id and password are optional, and provided for ease of use in development environments. In production environments, these values are usually encoded in the file

access layer, taken either from a secure configuration file, or hidden in some manner in the code.

Dfl Rules

Rule name:	When:	Meaning:
`clean ids`	(unused)	The database uses clean identifiers, except for tables where a specific primary index is defined.

The Table Item

This item defines a database table. It has these attributes:

Attribute:	Default:	Has this purpose:
`name`	(mandatory)	Specifies the name of the table, as created in the database. It must be unique within the database definition.
`description`	name	A description of the table item. This may be used when generating end-user messages that refer to the table, so it should be an good name for the table data.
`realname`	name	The real, physical name of the table. The default value for this attribute is the database.prefix followed by the table name.
`prefix`	dfl prefix	If specified, the database names of the table and all fields contained in the table are prefixed by these letters. You should use the prefix attribute to ensure that applications can use field names like ‘date’ and ‘time’ safely.

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

A table item must contain one or more field items.

Table Rules

Tables inherit all rules applied to group domains used in the table. Rules can also be applied directly to the table.

Rule name:	When:	Meaning:
`allocation`	*dbname*	For the database product 'dbname' specifies allocation information (storage requirements). 'dbname' is an implementation-defined condition. The rule value is used in an implementation-dependent manner.
`location`	*dbname*	For the database product 'dbname' specifies information about the table location. The rule value is used in an implementation-dependent manner.
`security`	*dbname*	For the database product 'dbname' specifies information about the table security. The rule value is used in an implementation-dependent manner.
`child only`	object	The table may only be used as a child table, i.e. attached to another table in an object. This rule can be applied to prevent inadvertent use of a table as an object root table.
`use workflow`	object	Objects based on the table are controlled by a multi-state workflow definition that is stored in the database.

The Table Field Item

This item defines a field in a database table. It has these attributes:

Attribute:	Default:	Has this purpose:
`name`	(optional)	Specifies the name of the field. If this is not specified, it defaults to the name of the domain. Group domains use the field name, if provided, as a prefix. If no domain is specified, the field name is mandatory.
`domain`	(optional)	Defines the way that the field is stored. A domain defines an abstracted data type for the field. The domain hides the specific size, type, and implementation of the field.
`realname`	name	The real, physical name of the field. The default value for this attribute is the dfl.prefix or table.prefix followed by the field name.
`type`	(mandatory unless a domain is specified)	Defines the field's data type, if no domain is specified. The valid data types are: textual, numeric, boolean, date, time, and timestamp.
`size`	(optional)	Defines the size of the field, in characters of storage space. This attribute is required for textual and numeric fields.
`decs`	0	Defines the number of decimals, for numeric fields only.

default	0 or ""	Defines the default value for the field, when a record is inserted with partial information.

The field item value is a one-line description of the field, if this is required. If this is not specified, it may be inherited from the domain description, if that is supplied.

Fields do not need to be defined with domains – the use of domains is appropriate to most cases, but for fields with a very specific type that is used only once, the extra effort of defining a domain may be excessive.

Domains like 'text2' meaning 'text field with 2 characters' are to be avoided, since they add complexity to the definition without benefit. It is better to define the field type and size directly.

Table Field Rules

Table fields inherit rules that are applied to their domain. These rules can also be applied directly to the table field. There is no way to disregard a rule that is applied to a domain, but it can be overridden by applying the same rule, with another value, to the field.

The Table Index Item

This item defines an index for a database table. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the index. If this is 'primary', it defines the table's primary index. If this is anything else, it defines an alternate index on the table. Primary indexes must be unique.
unique	0	This attribute is used only for alternate indexes. Primary indexes are always unique and the 'unique' attribute, if specified, must be '1'.

The index item value is a one-line description of the index, if this is required.

Each index name must be unique within the table. E.g. it is not permitted to define two primary indexes.

A index item must contain one or more field items.

Indexes do not need to be defined in all cases. If the database uses clean identifiers, all applications using the DFL can assume that the index is a single field called 'id' unless a primary index is specifically defined.

The Index Field Item

This item defines each field in a table index. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the field. The field must exist in the table that contains the index. The order of fields in an index is important. An index containing fields A and B is not the same as an index containing fields B and A.
order	ascending	If 'descending', this field is sorted in descending order.

The Link Item

This item defines relationships between tables. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(optional)	Specifies the name of the relationship. This is not necessary, but is useful for generating documentation.
type	(mandatory)	One of: "childof", "reference", "join", or "multiplex". We define parent-to-child links in the child table, to make it simpler to add and remove tables (only the child is changed, not the parent) for each child added. We define foreign-key references in the table that makes the reference. A join link specifies that the link can be followed in either direction, either as a 1-to-1 join, or as a parent-to-child link. Join links are typically used in tables that serve to create N-to-N relationships between tables.
table	(mandatory)	Name of table that is referenced by the link. For childof links, this is the parent link. For reference links, this is the reference table.

A link item must contain one or more field items.

A **childof** link implies that when the user is working with the parent, they can ask to work with the list of children attached to that parent. This is a 1-to-N link from the parent to the child table, where there may be zero or more children.

A **reference** link implies that when the user sees this field on the screen, it is shown as a decoded label, rather than as the real field value. For instance, instead of a country code, the user might see the translated country name. This is an optional 1-to-1 link from the current table to the referenced table.

A **join** link implies that the target table is an extension of or part of the current table. For instance, when showing a list of records in the current table, the joined table might be shown as part of the current table. This is an optional 1-to-1 link from the current table to the joined table.

A **multiplex** link is a reflexive link used to define relationship tables. The most typical use for this is to define a table that creates binary (N-to-N) relationship between two otherwise independent tables. Other uses for multiplex links are of dubious value although tertiary relationships may be useful in some cases. A multiplex link is equivalent to defining both a 'childof' link and a 'join' link.

The rationale for defining links in one table or another (e.g. 'childof' as opposed to 'contains') is the ease of maintenance of the DFL when adding or removing a table definition. Thus, since join links are typically used to implement tables that create N-to-N relationships by storing pairs of keys, they are specified in that table rather than in each parent.

When using clean identifiers, the source and target fields for links can be inferred. In other cases, the source and target fields can be defined using link <field> items.

Link Rules

Rule name:	When:	Meaning:
must exist	(unused)	For childof links, at least one child record must exist when the parent record exists. This rule is not implemented by the database access layer, but can be used by the workflow or user interface layers. For reference links, the reference may not be empty.

The Link Field Item

This item defines each field in a link item. These fields specify the values to be used in the current table. A link field item has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the source field. The field must exist in the table that contains the link. The order of fields in a link is important.
target	(optional)	Specifies the name of the field in the table referred to by the link. The default value for this depends on the type of the link and whether the database uses clean identifiers.

The Domain Item

This item defines a domain. It has these attributes:

Attribute:	Default:	Has this purpose:
name	(mandatory)	Specifies the name of the domain. It must be unique within the dfl definition.
type	(mandatory)	Defines the domain data type. The valid data types are: textual, numeric, boolean, date, time, timestamp, and group.
size	(optional)	Defines the size of the field, in characters of storage space. This attribute is required for textual and numeric fields.
decs	0	Defines the number of decimals, for numeric fields only.
default	0 or ""	Defines the default value for the field, when a record is inserted with partial information.

The domain item value is an optional one-line description of the domain.

Domains are shared by all tables in the dfl. A domain is used for several purposes:

- To specify data types in one single place, for instance a data type like 'id' can be defined in one place as a numeric field with 9 digits.

- To define data types that correspond to a set of values. The domain can specify the list of valid values, so that these can be used by all front-end programs that show data of this domain.
- To define structured (grouped) fields. For instance, 'address' may be defined as multiple individual fields, each with their own domain. This allows repeating structures to be isolated and normalised in the DFL.

DFL Abstracted Data Types

When domain (and field) items refer to a data type, they refer to an abstracted data type whose actual implementation depends on the database product being used. The mapping from the DFL data types to the database data types is handled by the DFL code generation module (mod_dfl).

We give an example of this mapping, here for a MS Access database accessed through ODBC:

DFL data type (size):	ODBC data type:
textual (1)	char
textual (2-255)	varchar (n)
textual (255-…)	longtext
boolean	byte
date	long (stored as YYYYMMDD)
time	long (stored as HHMMSSCC)
numeric (1-4)	integer
numeric (5-9)	long
numeric (10-…)	double
numeric with decimals	double
timestamp	datetime

Domain Rules

Rules applied to a group domain are inherited by any table that uses the domain. This can be useful for defining rules that apply to the table but depend on some group domain being present.

Rule name:	When:	Meaning:
show	detail, create, summary, grid, all, *	Define a presentation-layer attribute for the field when used in a specific type of screen. The detail, create, summary, grid, conditions refer to those specific types of screen. The all condition refers to all screens. You can use the "*" condition to apply the rule to two or more types of screen. Specify these as '<when>' children of the rule, e.g.: <rule name = "show" when = "*" what = "label" value = "Field label:" > <when>detail</when> <when>create</when> </rule>.
record id	insert	Field contains a unique record ID, which is a sequential value calculated when the record is inserted. This rule must be applied to primary index id fields when the database uses clean identifiers (it is not assumed).
set	insert	Field is forced to a specific value when the record is created. The value attribute specifies a constant value, which must match the type of the field.
not null	insert	Field cannot be zero or spaces. Timestamp fields take the current date and time as a default value. Date fields and time fields take the current date and time. For other types of field the value attribute provides the default value for the field.
user id	insert	Field contains the current user id. The DFL specifications assume that this information is available at the time that the record is created.
timestamp	insert	Field contains the time and date that the record is created.
set	update	See 'insert' above.
not null	update	See 'insert' above.
user id	update	See 'insert' above.
timestamp	update	Field contains the time and date that the record is updated.
set	delete	Field is forced to a specific value when a request is made to delete the record. When a table contains one or more fields with 'set when = "delete"', records in the table are not physically deleted from the database, but 'flagged' as such.
user id	delete	See 'insert' above.
timestamp	delete	Field contains the time and date that the record was flagged for delete.

The following table lists the user-interface rules allowed by the "show" rule:

What:	Meaning:
hidden	Field is hidden. By default, fields are shown to the user.
html	Field contains HTML text and must be encoded using an HTML encoding function before it can be shown on a browser page.
password	Field is a password.
attachment	Field is a file attachment (the actual field value is the full path to the file).
startyear	Field is a date with a specific year range. This argument defines the starting year, specified as a relative value (+n or –n). The default is –20.
endyear	Field is a date with a specific year range. This argument defines the starting year, specified as a relative value (+n or –n). The default is +10.
starthour	Field is a time with a specific hour range. This argument defines the starting hour, as an absolute 24-hour value.
endhour	Field is a time with a specific hour range. This argument defines the starting hour, as an absolute 24-hour value.
interval	Field is a time with a minute interval greater than one.
true	Field is a boolean field – this argument specifies the text to be shown when the field is true. The default value is "Yes".
false	Field is a boolean field – this argument specifies the text to be shown when the field is false. The default value is "No".
showsize	Field is a textual or numeric field – this argument specifies the size of the field on the screen. The default is equal to the field size.
rows	Field is a textbox field. This argument defines the vertical size of the textbox for screens where this applies.
cols	Field is a textbox field. This argument defines the horizontal size of the textbox for screens where this applies.
label	Defines the label used for the field.
select	Defines the display style type for a multi-value select field. The valid values for this argument are: "drop down", "radio", and "radio down".
case	Defines a case-conversion rule applied when the field is shown to the user, and accepted from the user. The valid values for this argument are: "upper", and "lower".
blank	Defines a blanking rule for numeric fields. The argument value "1" (the default) means that the field is shown blank when it contains zero, the argument "false" means the field is not shown blank.
join	Specifies that the field is shown on the same line as the previous field. This argument only makes sense in detail screens, where fields are normally shown one per line. The argument value is by default "1", you can specify "0" to override a join rule inherited from a domain.

The Domain Value Item

This item defines a domain value. It has these attributes:

Attribute:	Default:	Has this purpose:
key	(mandatory)	The value key. This is the value that is stored in the database. The types used for the value keys must match the domain type and size definition.
label	key	The value label. If this is not provided, the label takes the value of the key. The label is shown to the user in place of the key.

When a domain has one or more <value> items, it specifies that all fields using this domain are shown as drop-down select boxes, or radio options. Value items can also be attached to fields. If they are attached both to a field and its domain, the domain values are appended to the list defined in the field. Value selection lists are always ordered by key.

The Domain Field Item

When a domain has the type 'group', it must contain one or more <field> items. These fields have the same set of attributes as table fields, and can themselves refer to group domains, to as many levels as is wanted. The original field name is used as a prefix for the group fields. If a table contains a field with name 'user', and domain 'address', and the address domain is itself defined as two fields, 'street' and 'city', the result is that two fields are defined in the table: 'userstreet' and 'usercity'. The original field name can be omitted.

The Domain Link Item

A group domain can also include links, which are inherited by all tables that use the group in one or more places. Links specified in groups must always specify the field(s) that they use.

The Domain Link Field Item

The domain link <field> item has one attribute, 'name'.

The Inherit Item

Not currently implemented.

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

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

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

The Include Item

This item defines a DFL subfile that is included into the current DFL. 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 DFL file must conform to certain rules. Firstly, it does not contain a 'dfl' 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 DFL. For example, two domains 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 <dfl> item.

Worked Example

...