ARCA

Libraries Programme Project LIB-ARCA/2-3039

Title : ARCA Target Configuration Interface Design Document

Document Reference : ARCA/T141/TID

Version : Version 1.0

Date : 26 May 1997

Author(s) : Maria Bruna Baldacci, Donatella Castelli, Pasquale Pagano

Distribution : Intecs Sistemi
CNR - IEI
SABINI
Fundacion Sancho El Sabio
Universita' di Pisa
Regione Toscana

Abstract : This document describes the mechanisms of the interface used to configure the ARCA target upon a specific OPAC

Keywords :

Reviewed by :

Approved by :

Document Status Sheet

Issue	Changes	Date	Reason
1.0	New document	26 May 1997	

Table of Contents

1. Introduction

This document describes the design phase of the Arca Target Configuration Interface (ATI). This application provides a graphical interface to assist the OPAC administrator in configuring the ARCA Target system according the underlying OPAC. The loading task is simplified by both decomposing this activity into separate working sections and providing window environment. Moreover, the Interface guarantees the consistency of the input data by providing basic operations that always preserve their consistency. The output is a set of files that are used by the ARCA Target system to initialize the Dictionary which store the characteristics of the underlying OPAC.

This document is organized as follows:

- ATI specification: brief analysis of the inherent problems regarding the set up of an Arca Target;
- ATI architecture: development environment and design of the application.

2. Target Interface Specification

To be able to connect different types of OPACs, the ARCA Target stores the characteristics of individual OPACs into its Dictionary [ARCA/T12/ADD, ARCA/T13/DSD].

The aim of ATI is to provide an automatic environment that assists the OPAC administrator in loading and updating the information required to initialize the ARCA Dictionary upon a specific OPAC.

Let us recall that the structure of the Dictionary resembles that of the Z39.50 EXPLAIN database [ANSI/NISO, 1995]. However, the information stored by the Dictionary is not exactly the same as those of the EXPLAIN database since the Dictionary contains both additional and missing information with respect to the Explain database, and vice versa.

The structuring of the Dictionary is as follows (see also figure 2.1) :

1. *TargetInfo*

   Lists information about the connection to the target;

2. *DatabaseInfo*

   Lists information about the existing databases;

3. *TermListInfo*

   Reports information concerning the term lists supported;

4. *AttributeSetInfo*

   Includes information about the attributes for the databases;

5. *RecordSyntaxInfo*

   Lists information about the structure of the record in the databases;

6. *ElementSetDetails*

   Lists information about the available presentation formats.

A detailed description of each of the above categories is given in Appendix A.

Figure 2.1 highlights the relationships between Dictionary categories.

The illustrated structureness of the Dictionary expresses inherently referential constraints among the given categories. These are graphically expressed as arrows between the boxes. For example, the arrow that originates from Database name within the *TargetInfo* box towards the *DatabaseInfo* box expresses that for each

Database name there must be a *DatabaseInfo* element that contains information about that database.

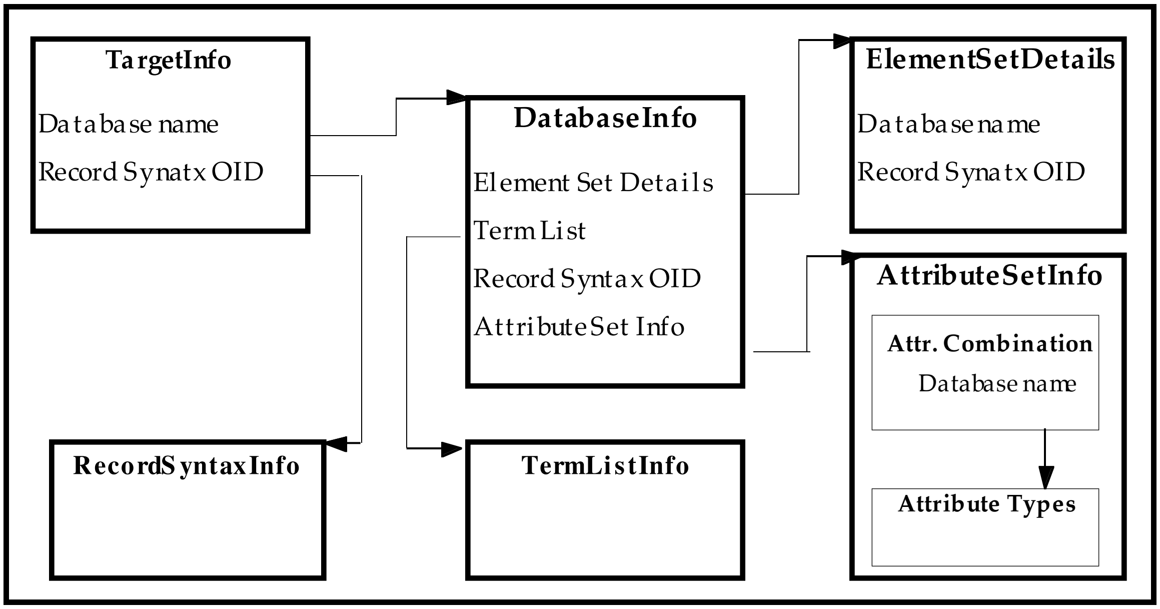

Figure 2.1 : The relationships between Dictionary categories

Other constraints on the Dictionary fields, not graphically shown, are:

- Type constraints - the possible types are numeric or alphanumeric,
- Mandatory constraints - a field cannot be empty.

The operations provided by ATI to load the Dictionary are defined so as to automatically preserve, as much as possible, these constraints. When this is not possible, these operations check the constraints and fail when they are violated. To achieve this goal the interface provides, for each category, a set of specific operations that are the only ones that can be invoked to operate on that category. Moreover, it restricts the possible sequences of calls to these operations by imposing an ordering on the visiting of the categories. It also dinamically decides the information to ask for, according to the information that was inserted beforehand. In particular, the rules below hold (in the following, we will call "section" the set of operations applicable on a specific category).

- In the *TargetInfo* section the database names and the record syntax OIDs are mandatory since they are the access keys to other sections. A change or removal of a value in this field can be done only by the operations associated to the TargetInfo section. These operations automatically update the rest of the Dictionary thus preserving its consistency. A change of a database name in the Target section, for example, causes a similar update in the DatabaseInfo, ElementSetDetail and AttributeInfo sections.

- In the *DatabaseInfo* section, the names of the term lists associated with the described database, the supported attribute combinations, the record syntaxes and the element sets are specified. The record syntaxes must be a subset of those listed in the TargetInfo section. This constraint is preserved by, firstly assuring that the Record Syntax OIDs in TargetInfo be specified before the loading of the Record Syntax OIDs in DatabaseInfo occurs. The set of the OIDs specified beforehand can then be used to restrict the set of admissible values for the next loading.

- In the *ElementSetDetails* section a record syntax for the associated database is specified. Obviously, this value must be among those listed for that database. Similarly to the previous situation, this constraint is preserved by assuring that the DatabaseInfo section be specified before the ElementSetDetails can be filled and then restricting the possible values for the record syntax OIDs accordingly.

- In the *AttributeSetInfo* section for each combination of attributes inserted, it is necessary to define the database that supports this combination: also in this case the choice is limited only to those databases that in the previous sections supported the attribute set in question. Moreover, in order to preserve the consistency among the attribute combinations and the attribute types specified in this section, the update operations for these two fields are also ordered. This enables to automatically identify the attribute types and the relative attribute values for which a description has to be supplied.

Some fields of the Dictionary are mandatory only if other fields have certain values. These kind of constraints are preserved by dynamically configuring the interface according to such values. For example, in the TargetInfo section the fields <RPN to string conversion>, <native AND>, <native OR>, <native AND-NOT>, <native PROXIMITY> must be specified only if the field <RPN natively support> has the value “No”.

The violation of a constraint that cannot be automatically guaranteed results in the failure of the operation that has broken the consistency. As a result, the interface is able to guarantee that no inconsistent configuration for the Dictionary will ever be loaded.

Therefore ATI provides an automating supporting environment for easy loading of the Dictionary and enables the OPAC Administrator to overcome the flat and redundant structure of the Dictionary categories records. Moreover, due to the guided loading and navigation and to the checks on the integrity, storing of inconsistent information is prevented.

3. Target Interface Architecture

This section has a twofold purpose. Its primary purpose is to present the design of the ATI application. But it also has the purpose of presenting the context in which the design decisions were taken and to provide the rationale of those decisions.

We have raised the design level of ATI application to the Tcl/Tk scripting level.

In order to avoid drowning the reader in technical details, we have organized the documentation of the design phase into three parts:

- Tcl/Tk : a modern user interface design technology;
- ATI design;
- The purest, code-level details where complete scenarios and code are presented.

3.1 Tcl/Tk : a modern user interface design technology

Tcl stands for Tool Command Language. It is a scripting language, in the sense of the Unix shell, and was originally developed for the Unix system but it can also be ported to several of the most important host platforms, including Unix-like Windows and Macintosh.

Tcl programs are easy to write, especially easy to debug and can be dynamically modified, but the strongest points of Tcl are its powerful facilities for handling certain kinds of specialized tasks and data structures:

- associative array, whose dimension is not known a priori and which does not have integer indices;
- list handling and string manipulation, for which Tcl provides many powerful facilities.

Tk (ToolKit) is an extension to Tcl. It provides facilities for creating a graphical user interface based on the X Window system.

Tcl/Tk constitues, therefore, an excellent support for graphical user interface that should manage and manipulate text files.

The programs are event-driven in the sense that the window management system notifies the application of events (movements of mouse, input from the mouse, the keybord or the network) as they occur, and the application takes appropriate action. In practice, individual procedure, called event handler, is associated to each type of event. This procedure is called directly by the window management system. By its very nature, in a Tcl/Tk application there is no real central flow, but an initialization script and a set of event handlers. The elegance and power of Tcl as an interpreted language is evident here: event handlers can be created even at runtime, e.g. by other event handlers.

3.2 Target Interface Design

In this section we present the interface architecture using a descriptive approach. This approach is motivated by two factors:

- the nature of event-driven applications is not suitable to a formal description of the interactions between the diverse modules, components of the application itself, in that it is the user who determines the invocation sequence of the modules;
- the readability of the document increases if we put into evidence the structure of the interface in terms of a set of functionalities without entering into details of the their single event handlers.

The picture below outlines the interactions between the different functionalities of the interface and highlights the principal components of each single functionality.

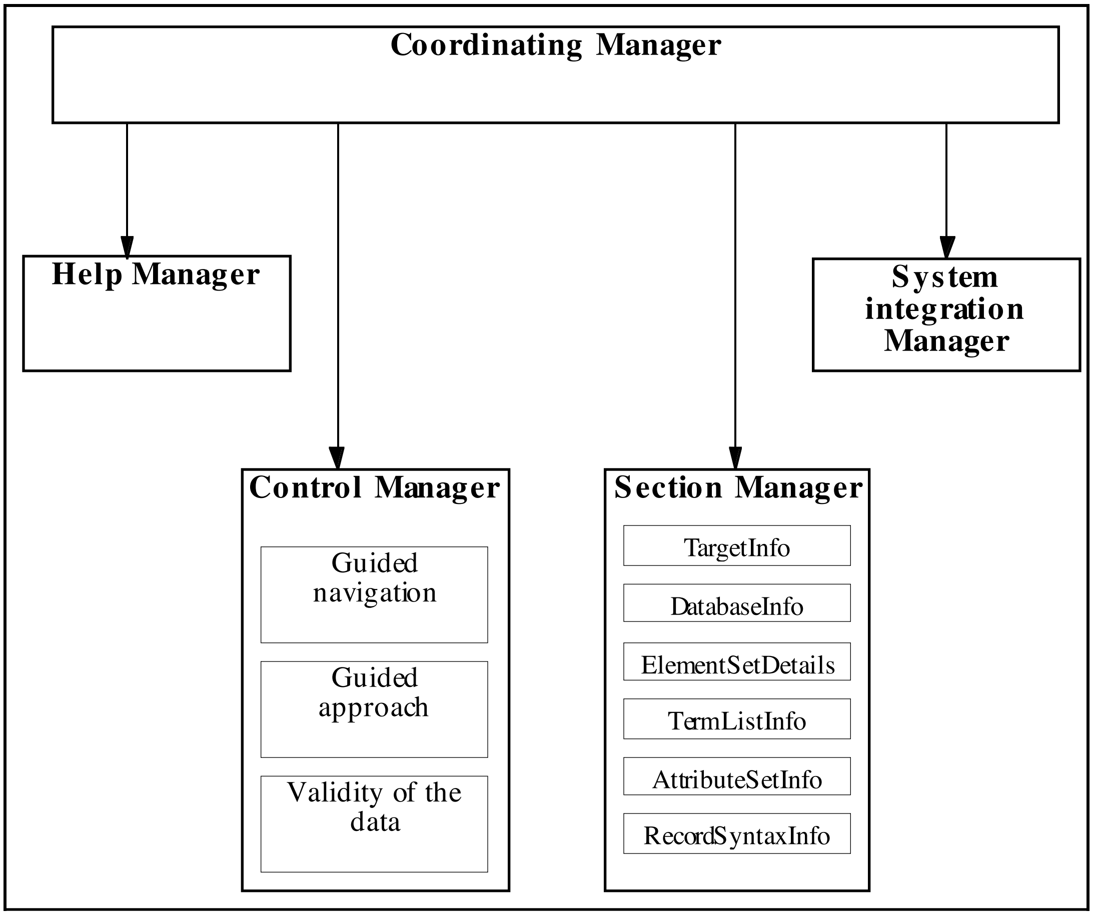

Figure 4.1 The interface architecture

We will now analyze in detail the different components represented so as to highlight their properties.

3.2.1 Coordinating Manager

Technically the coordinating manager is made up of two logical units:

- the window management system, obviously external to the application;
- a set of event handlers that are invoked by the first unit and that coordinate other event handlers when necessary.

Given the inherent nature of a Tcl/Tk application, the management of the various events is delegated to the window management system that, as stated earlier, invokes an opportune event handler each time it receives input from the outside. Those tasks relative to the activation of the interface and to the maintenance of the system in a consistent state, instead, are carried out by the coordinating manager module. This latter functionality is required in that not all the functions need necessarily be employed by the users and it is the job of the coordinator to disactivate or activate the suitable functions according to the choices made by the user.

3.2.2 Help Manager

On-line help is provided for each state of the system and for each situation in which the user may have a need. Each window application has a Help key that opens a window of the Help system visualizing the appropriate Help page.

Two types of on-line help are available that for each section visualize:

- a list of the mandatory fields of the current section;
- the meaning of each entry of the section. This explanation is supplied using the description contained in the official standard ANSI/NISO Z39.50.

Further pages of the Help System become visible each time the user inserts incorrect data.

Hence the user has access to a full help that permits the correct initialization of the Dictionary. At the same time the user benefits from a useful on-line debugger that supplies all the necessary information to remove those situations where an error has been caused by a type- or completeness constraint.

3.2.3 System Integration Manager

System Integration Manager comprises all those procedures that manage the saving of information, cancellation of the modifications on the initial files and the preparation of the Dictionary files.

Let us now analyze in detail the single functionalities.

Saving

The saving of information occurs only at the end of each section when the system has checked that the configuration given by the user is correct.

Cancelling

Regarding the second functionality carried out by the System Integration Manager, we must point out that the user must always have the possibility of cancelling the operation he or she is performing without compromizing the work already done. For this reason several levels of "undo" have been individuated:

- entry: it is always possible to cancel a value inserted in a entry by simply cancelling the contents of the fields;

- window: the updates on a window can be removed by restoring the previous state;

- section: it is always possible to cancel the updates introduced in a section by restoring the previous state. This operation removes all the modifications carried out within the section, and therefore "undoes" all the updates introduced in all subsections of the current section;

- initialization: the entire work section can be cancelled. This operation repeats the preceding operation for each section, thereby restoring the last saved configuration.

Preparing the files of the Dictionary

At the end of all the sections, and only then, the files of the Dictionary are automatically generated. Proceeding in this way, in fact, we can be certain that:

- those files are without errors of a syntactic nature since they are generated automatically by the application;

- the information contained in those files are correctly correlated, thus respecting the obligations imposed by the existing relations within the Dictionary (see Figure 2.1).

3.2.4 Control Manager

The Control Manager groups together all the functionalities relative to the control of the data inserted. Its job, therefore, is to guarantee the absence of errors of a syntactic nature, respect the relations between the various categories and insert all of the necessary information. Details of the control are provided in Chapter 2.

3.2.5 Section Manager

Taken together, the procedures that manage the data entry for each single section are coordinated by the functionalities that permit the correct sequentiality between the dialog boxes .

4. References

[ANSI/NISO, 1995] *Information Retrieval: Application Service Definition and Protocol Specification. ANSI/NISO Z39.50-1995. Available: ftp.loc.gov .*

[ARCA/T13/DSD] *ARCA Dictionary Schema Document, ARCA/T13/DSD, February 1996*

[ARCA/T12/ADD] *ARCA Design Document, ARCA/T12/ADD, November 1995.*

Appendice A: Detailed description of the Dictionary Categories

Target Information

```
TargetInfo:
common_info:
        dateAdded:
                value=String_Value;
        dateChanged:
                value=String_Value;
        expiry:
                value=String_Value;
        humanStringLanguage:
                value=String_Value;
target_name:        // Mandatory
        value=ARCA;
news:
        value=Text_Value;
welcome_message:
        value=Text_Value;
target_descr:
        value=Text_Value;
usage_restriction:
        value=Text_Value;
payment_address:
        value=Text_Value;
hours:
        value=String_Value;
result_set_naming:          // Mandatory
        value=(Y|N) ;
multi_database_search:   // Mandatory
  support :
        value=(Y|N) ;
mode:
        value=(ARCA|native);
max_result_set:     // Mandatory
        value=Int_Value;
max_num_term:   // Mandatory
        value=Int_Value;
max_set_size:       // Mandatory
        value=Int_Value;
preferred_message_size:          // Mandatory
        value=Int_Value;
contact_info:
        value=Text_Value;
diagnostic_set:
        value=(1.2.840.10003.4.1);
selector_position:   // Mandatory
        value=(RIGHT|LEFT|ANY);
supported_authentication:        // Mandatory
        support:
                value=(Y|N);
        mode:
                value=(ARCA|native);
databases:     // Mandatory
        value=String_Value,
        ...................,
        ...................,
        value=String_Value.
record_syntaxes:    // Mandatory
        value=Oid_Value,
        ...................,
        ...................,
        value=Oid_Value.
```

Database Information

// The following description is repeated for each database supported

```
DatabaseInfo:         // Mandatory
common_info:
        dateAdded:
                value=String_Value;
        dateChanged:
                value=String_Value;
        expiry:
                value=String_Value;
        humanStringLanguage:
                value=String_Value;
database_name:     // Mandatory
        value=String_Value;
database_descr:
        value=Text_Value;
explain_database:
        support:
                value=(Y|N),
        mode:
                value=(ARCA|native);
user_fee:      // Mandatory
        value=(Y|N);
```

```
available:      // Mandatory
        value=(Y|N);
disclaimers:
        value=Text_Value;
news:
        value=Text_Value;
record_count:
        value=Int_Value;
def_order:
        value=Text_Value;
average_rec_size:
        value=Int_Value;
max_rec_size:
        value=Int_Value;
hours:
        value=String_Value;
best_time:
        value=String_Value;
last_update:
        value=String_Value;
update_interval:
        value=String_Value;
coverage:
        value=String_Value;
proprietary;
        value=(Y|N);
copyright_text;
        value=Text_Value;
copyright_notice:
        value=Text_Value;
producer_contact_info:
        value=Text_Value;
supplier_contact_info:
        value=Text_Value;
submission_contact_info:
        value=Text_Value;
title_string:
        value=Text_Value;
database_keywords:
        value=String_Value,
        ...................,
        ...................,
        value=String_Value.
RPN_query_Syntax:
        value=File;
query_types_supported:  // list of the query type
        private_capabilities:
```

```
        private_operators:
                operator:     //list of operators and descr
                        value=String_Value;
                description:
                        value=String_value;
                ...................;
                ....................;
                operator:
                        value=String_Value;
                description:
                        value=String_value.
        private_search_keys:
                search:       //list of search keys and descr
                        value=String_Value;
                description:
                        value=String_value;
                ....................;
                ....................;
                search:
                        value=String_Value;
                description:
                        value=String_value.
Rpn_capabilities:
        operators:
                value=Int_Value,
                ...............,
                value=Int_Value.
        result_set_as_operand_supported:
                support:
                        value=(Y|N),
                mode:
                        value=(ARCA|native);
        restriction_operand_supported:
                value=N;
        proximity:
                value=(Y|N),
                mode=(native)
iso8777_capabilities:
        search_keys:          //list of search keys and descr
                search:
                        value=String_Value;
                description:
                        value=String_value;
                ....................;
                ...................;
                search:
                        value=String_Value;
```

```
                    description:
attribute_sets:         // Mandatory
        value=Oid_Value,
        ..............,
        value=Oid_Value.
term_lists:    // Mandatory
        value=String_Value,
        .................,
        value=String_Value.
element_set_details:        // Mandatory
        value=String_Value,
        .................,
        value=String_Value.
searchable_with_databases:
        value=String_Value,
        .................,
        value=String_Value.
record_syntaxes:    // Mandatory
        value=Oid_Value,
        .................,
        value=Oid_Value.
```

Term List Information

```
// The following description is repeated for each tem list
// supported

TermList:
term_list_name:    // Mandatory
        value=String_Value;
title:
        value=String_Value;
search_cost:
        value=(optimized|normal|expensive|filter);
scannable:    // Mandatory
        support:
                value=(Y|N),
        mode:
                value=(ARCA|native);
broader:
        value=String_Value,
        .................,
        value=String_Value.
narrower:
        value=String_Value,
        .................,
        value=String_Value.
```

Attribute Set Information

```
// The following description is repeated for each
// attribute set supported

AttributeSetInfo:
common_info:
        dateAdded:
                value=String_Value;
        dateChanged:
                value=String_Value;
        expiry:
                value=String_Value;
        humanStringLanguage:
                value=String_Value;
attribute_set_oid:  // Mandatory
        value=Oid_Value;
attribute_set_name:
        value=String_Value;
// The following description is repeated for each
// attribute combination
attribute_combinations:  //begin repeatable item
        infix_word_list_operator:
                value=(String_Value,  native);
        infix_tags_operator:
                value=(String_Value,  native);
        combination:
                value=(attribute_type_Int_Value,  attribute_value_Int_Value)  ...
                        (attribute_type_Int_Value,  attribute_value_Int_Value);
        translation:
                value=String_Value;
        usable_with_database:
                value=String_Value;
        use_attributes:
                use_value:
                        value=Int_Value;
                tags:
                        value=String_Value,
                        ...................
                        value=String_Value.
                ......................
                use_value:
                        value=Int_Value;
                tags:
                        value=String_Value,
                        ...................
                        value=String_Value.      //end repeatable item
.........................
attribute_combinations:  //begin repeatable item
.........................
.........................         //end repeatable item
attribute_types:
        attribute_type:       //begin repeatable item
                value=Int_Value;
        attribute_name:
                value=String_Value;
        attribute_descr:
                value=String_Value;
        default_value:
                value=Int_Value;
        attribute_values:   //begin repeatable item
                name:
                        value=String_Value;
                value:
                        value=Int_Value;
        is_selector:
                value=(Y|N);         //end repeatable item
        .........................
        attribute_values:   //begin repeatable item
                name:
                        value=String_Value;
                value:
                        value=Int_Value;
                is_selector:
                        value=(Y|N);         //end repeatable items
...........................
attribute_types:
        attribute_type:       //begin repeatable item
                value=Int_Value;
        attribute_name:
                value=String_Value;
        attribute_descr:
                value=String_Value;
        default_value:
                value=Int_Value;
        attribute_values:   //begin repeatable item
                name:
                        value=String_Value;
                value:
                        value=Int_Value;
                is_selector:
                        value=(Y|N);         //end repeatable item
        ........................
        attribute_values:   //begin repeatable item
                name:
                        value=String_Value;
                value:
                        value=Int_Value;
                is_selector:
                        value=(Y|N);        //end repeatable items
```

Record Syntax Information

```
// The following description is repeated for each record syntax
// supported

RecordSyntaxInfo:
common_info:
        dateAdded:
                value=String_Value;
        dateChanged:
                value=String_Value;
        expiry:
                value=String_Value;
        humanStringLanguage:
                value=String_Value;
recordsyntax_name:
        value=String_Value;
recordsyntax_oid:  // Mandatory
        value=Oid_Value;
recordsyntax_descr:
        value=Text_Value;
is_default:
        value=(Y|N);
ASN_1_syntax:
        value=Text_Value;
```

Element Set Details

```
// The following description is repeated for each element set
// supported

ElementSetDetails:          // Mandatory
common_info:
        dateAdded:
                value=String_Value;
        dateChanged:
                value=String_Value;
        expiry:
                value=String_Value;
        humanStringLanguage:
                value=String_Value;
element_set_name:       // Mandatory
        value=String_Value;
database_name:    // Mandatory
        value=String_Value;
record_syntax_oid:      // Mandatory
        value=Oid_Value;
elementset_descr:
                value=Text_Value;
is_default:
        value=(Y|N);
element_set_details:
        perElementDetails:       //begin repeatable item perElementDetails
                name:
                        value=String_Value;
                repeatable:
                        value=(Y|N);
                required:
                        value=(Y|N);  //end rpeatable item perElementDetails
        perElementDetails:       //begin rpeatable item
                ...............;     // end rpeatable item
```