Accessing Data in a Tamino Database

This chapter describes the different aspects of accessing a Tamino database with the Natural data manipulation language (DML).

The following topics are covered:

- Prerequisite
- DDM and View Definitions with Natural for Tamino
- Natural Statements for Tamino Database Access
- Natural for Tamino Restrictions

For information about how to configure Natural to work with Tamino, see *Natural for Tamino* in the *Database Management System Interfaces* documentation.

Prerequisite

Tamino stores structured data-oriented XML documents in containers called doctypes. The doctypes are grouped logically together in so-called collections. Collections are stored in a Tamino database, which is the physical container of data.

The kind of data that can be stored in Tamino and that is to be accessed by Natural for Tamino must be defined in a Tamino XML Schema.

DDM and View Definitions with Natural for Tamino

This section describes the basic concepts of the Tamino XML schema language, Natural DDMs and view definitions and how they interact with Natural for Tamino.

The following topics are covered:

- Introducing Tamino XML Schema Language
- DDMs from Tamino
- Arrays in DDMs from Tamino
- Example of a DDM
- Definition of Views

Introducing Tamino XML Schema Language

The Tamino XML schema language is used to define a data type-oriented description of the structure of XML documents. In Tamino, a doctype represents a container for XML documents with the same root element and the same structure within a collection.

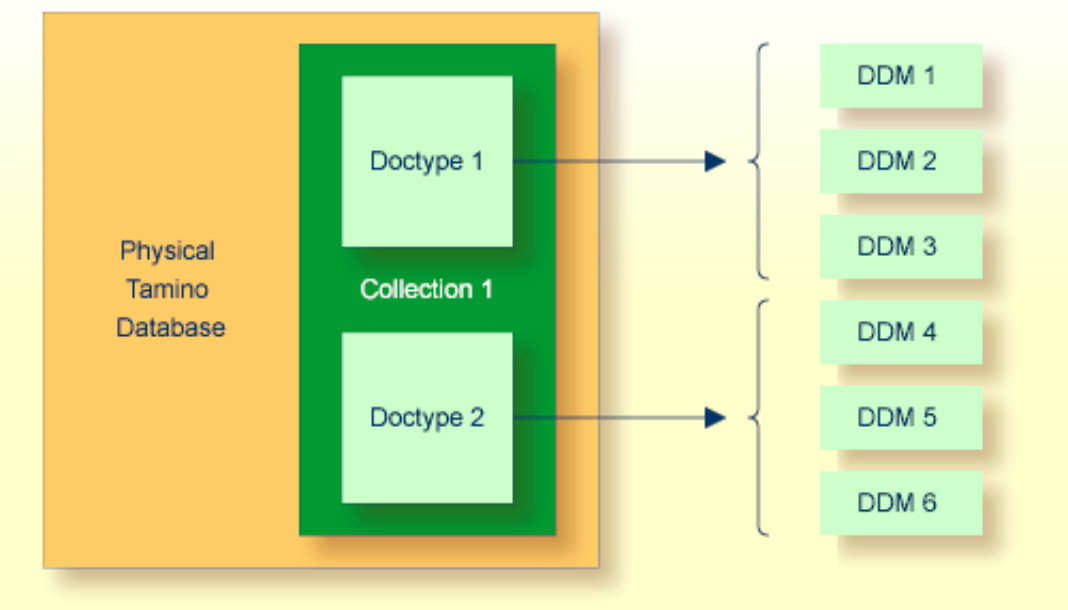
In Tamino, a collection is a container for a set of varying doctypes, so that a collection can be seen as the logical grouping of doctypes that belong together.

In a Tamino XML schema definition, a doctype is defined together with the collection in which it is contained. One Tamino XML schema can define more than one doctype and it can also define doctypes for more than one collection.

For more information on the Tamino XML schema language, refer to the Tamino documentation.

DDMs from Tamino

For Natural to be able to access a Tamino database, a logical connection between a Tamino doctype and the Natural data structures must be provided. Such a logical connection is called a DDM (data definition module).



A Natural DDM generated from a Tamino database is a representation of one doctype defined in one schema. The DDM contains information about the type of each data field and all the necessary structural information as defined in the corresponding Tamino XML schema. To generate a new DDM, the doctype must be selected from a list of all doctypes available in a given collection. Since one collection is bound to one Natural database ID (DBID), it is necessary to use a second DBID if a doctype from another collection is to be accessed.

A Tamino XML schema describes data and data structures in a very different way than with Natural data definitions. Therefore, specific mappings are introduced to derive a Natural data format from a Tamino XML schema data type.

You define DDMs with Natural DDM Services. For more information about Tamino XML schema mapping, refer to *Data Conversion for Tamino* in the *DDM Services* section of the *Editors* documentation.

For the field attributes defined in a DDM, refer to the *DDM Editor*, *Using the DDM Editor WindowDDM Services*, *Using the DDM Editor Screen* section in the *Editors* documentation.

Arrays in DDMs from Tamino

If you define an XML element with a `maxOccurs` value greater than one in the Tamino XML Schema, then this element can occur as often as this value indicates. Such a construction is mapped either on a Natural static array definition or on a Natural X-Array definition. Depending on the type of the XML element you are dealing with, the following situations may occur:

- If the XML element is a `complexType` with `complexContent` (i.e. it is an element containing other elements) then the generated corresponding Natural group will be an indexed group.
- If the XML element is a `simpleType` (i.e. the element is holding data only) or a `complexType` with `simpleContent` (i.e. the element has only data and attributes but no other elements) then the generated Natural data field will be an array.

For further information about mapping `maxOccurs` definitions onto Natural arrays, see *Data Conversion for Tamino* in the *DDM Services* section of the *Editors* documentation. The array boundaries or the kind of the array (static array or X-Array) can be adapted in a corresponding view definition as usual.

Example of a DDM

This is an example of an `EMPLOYEES` DDM generated from a Tamino XML Schema definition.

The schema can, for example, be defined with the Natural demo application `SYSEXDB`:

```
DB: 00250 FILE: 00001  - EMPLOYEES-XML
TYPE: XML
COLLECTION: NATDemoData
SCHEMA: Employee
DOCTYPE: Employee
NAMESPACE-PREFIX: xs
NAMESPACE-URI: http://www.w3.org/2001/XMLSchema
T L  Name                            F Leng       D Remark
- -- ------------------------------- - ---------- - -----------
G  1 EMPLOYEE
     FLAGS=MULT_REQUIRED,MULT_ONCE
     TAG=Employee
     XPATH=/Employee
G  2 GROUP$1
     FLAGS=GROUP_ATTRIBUTES
   3 PERSONNEL-ID                    A          8 D xs:string
     FLAGS=ATTR_REQUIRED
     TAG=@Personnel-ID
     XPATH=/Employee/@Personnel-ID
G  2 GROUP$2
     FLAGS=GROUP_SEQUENCE,MULT_REQUIRED,MULT_ONCE
G  3 FULL-NAME
     FLAGS=MULT_OPTIONAL
     TAG=Full-Name
     XPATH=/Employee/Full-Name
G  4 GROUP$3
     FLAGS=GROUP_SEQUENCE,MULT_REQUIRED,MULT_ONCE
   5 FIRST-NAME                      A         20 D xs:string
     FLAGS=MULT_OPTIONAL
     TAG=First-Name
     XPATH=/Employee/Full-Name/First-Name
   5 MIDDLE-NAME                     A         20 D xs:string
     FLAGS=MULT_OPTIONAL
     TAG=Middle-Name
```

```
    XPATH=/Employee/Full-Name/Middle-Name
  5 MIDDLE-I                      A         20 D xs:string
    FLAGS=MULT_OPTIONAL
    TAG=Middle-I
    XPATH=/Employee/Full-Name/Middle-I
  5 NAME                          A         20 D xs:string
    FLAGS=MULT_OPTIONAL
    TAG=Name
    XPATH=/Employee/Full-Name/Name
      . . .
  3 LANG                          A          3   xs:string
    FLAGS=ARRAY,MULT_OPTIONAL
    OCC=1:4
    TAG=Lang
    XPATH=/Employee/Lang
```

Definition of Views

In order to work with Tamino database fields in a Natural program, you must specify the required fields of the DDM in a Natural *view-definition* (see the `DEFINE DATA` statement). Normally, a view is a special subset of the complete data structure as defined in the DDM.

Tamino XML Schema->Natural for Tamino DDM->Natural *view-definition*

If the view is used to store XML objects, it has to contain all fields that are required to a generate documents that are valid according to the corresponding Tamino XML schema definition.

A view for the `EMPLOYEES-XML` DDM, where one of the view fields is a static array, might look like this:

```
DEFINE DATA LOCAL
01 VW VIEW OF EMPLOYEES-XML
02 NAME
02 CITY
02 LANG (1:4)
END-DEFINE
```

Natural Statements for Tamino Database Access

The Natural DML statements which are provided for Tamino access can be subdivided into two categories:

- pure retrieval statements;
- database modification statements.

The Natural system variable `*ISN` is mapped on the Tamino `ino:id`.

Natural for Tamino Retrieval Statements

The following Natural statements can be used for database retrieval:

- `FIND`

This statement is used to select those records from a database which meet a specified search criterion.

- `GET`

  This statement is used to select one special record with its unique id from the database.

- `READ`

  This statement is used to select a range of records from a database in a specified sequence.

Not all of the possible options and all of the possible clauses of the retrieval statements can be used for Tamino access. Please read the appropriate section in the *Statements* documentation for a detailed description.

All statements are internally realized with the Tamino `_xquery` command verb. Statement clauses are mapped to corresponding Tamino XQuery expressions, e.g. search criteria are mapped to Tamino XQuery comparison expressions, sequence specifications are mapped to Tamino XQuery ordering expressions with sort direction.

The result set for the `FIND` and `READ` statements is determined at start of the loop and remains unchanged throughout the loop.

The following is an example of reading a set of employee records from a Tamino database where one view field is an array:

```
* READ 5 RECORDS DESCENDING CONTAINING A
* STATIC ARRAY IN THE VIEW DEFINE DATA LOCAL
01 VW VIEW OF EMPLOYEES-TAMINO
02 NAME
02 CITY
02 LANG (1:4)
END-DEFINE
*
READ(5) VW DESCENDING BY NAME = 'MAYER'
  DISPLAY NAME CITY LANG(*)
END-READ
*
END
```

Natural for Tamino Database Modification Statements

The following database modification statements are provided for use with Natural for Tamino:

- `STORE`

  This statement is used for inserting a new XML document into the database.

- `DELETE`

  This statement is used for deleting a document from the database. The `DELETE` statement implements a positioned delete.

For a detailed description of the statements, see the appropriate sections of the *Statements* documentation.

The DELETE statement is internally realized with the Tamino _delete command verb using the current ino:id, and the STORE statement is implemented with the _process command verb.

Example:

The following example program stores a new employee record with some data in the database:

```
* STORE NEW EMPLOYEE
DEFINE DATA LOCAL
01 VW VIEW OF EMPLOYEES-TAMINO
02 PERSONNEL-ID
02 NAME
02 CITY
02 LANG (1:3)
END-DEFINE
*
* FILL VIEW
PERSONNEL-ID := ’1230815’
NAME         := ’KENT’
CITY         := ’ROME’
LANG(1)      := ’ENG’
LANG(2)      := ’GER’
LANG(3)      := ’SPA’
*
* STORE VIEW
STORE RECORD IN VW
*
COMMIT
*
END
```

If the Tamino XML Schema defines data structures for a doctype as being mandatory, then these data structures must also be filled in the view before a STORE statement is issued, otherwise this will result in a Tamino error.

Natural for Tamino Logical Transaction Handling

Natural performs database modification operations based on transactions, which means that all database modification requests are processed in logical transaction units. A logical transaction is the smallest unit of work (as defined by you) which must be performed in its entirety to ensure that the information contained in the database is logically consistent.

A logical transaction may consist of one or more modification statements (DELETE, STORE) involving one or more doctypes in the database. A logical transaction may also span multiple Natural programs.

A logical transaction begins when a database modification statement is issued. Natural does this automatically. For example, if a FIND loop contains a DELETE statement. The end of a logical transaction is determined by an END TRANSACTION statement in the program. This statement ensures that all modifications within the transaction have been successfully applied.

Natural for Tamino Error Handling

In addition to Natural’s standard error messages there are two special error codes which provide additional information via a sub-error code.

Error Message NAT8400

```
NAT8400 Tamino error ... occurred
```

For this special error an additional sub-code number is shown. This number refers to a Tamino error message. Please see the Tamino *Messages and Codes* documentation. The user exit USR6007 in library SYSEXT is provided for obtaining diagnostic information in case a NAT8400 error occurs.

Here is an example of usage:

```
DEFINE DATA LOCAL
 01 VW VIEW OF EMPLOYEES-TAMINO
  02 NAME
  02 CITY
 01 TAMINO_PARMS
  02 TAMINO_ERROR_NUM        (I4)   /* Error number of Tamino error
  02 TAMINO_ERROR_TEXT       (A70)  /* Tamino error text
  02 TAMINO_ERROR_LINE       (A253) /* Tamino error message line
END-DEFINE
*
NAME := 'MEYER'
CITY := 'BOSTON'
STORE VW
*
ON ERROR
 IF *ERROR EQ 8400     /* in case of error 8400 obtain diagnostic information
  CALLNAT 'USR6007N' TAMINO_PARMS
  PRINT 'Error 8400 occurred:'
  PRINT 'Error Number:' TAMINO_ERROR_NUM
  PRINT 'Error Text  :' TAMINO_ERROR_TEXT
  PRINT 'Error Line  :' TAMINO_ERROR_LINE
 END-IF
END-ERROR
*
END
```

Error Message NAT8411

```
NAT8411 HTTP request failed with response code...
```

The error code from the HTTP server is delivered as additional information. See also REQUEST DOCUMENT statement, *Overview of Response Numbers for HTTP/HTTPs Requests*.

Example of Natural for Tamino Interacting with a SQL Database

This is a more sophisticated example of Natural for Tamino interacting with an SQL database; it retrieves data from a Tamino database and inserts or updates the corresponding row in an appropriate table in a SQL database.

```
*
* TAMINO DB --> SQL RDBMS EXAMPLE
*
DEFINE DATA LOCAL
* DEFINE VIEW FOR TAMINO
01 VW-TAMINO VIEW OF EMPLOYEES-TAMINO
02 PERSONNEL-ID
02 NAME
02 CITY
* DEFINE VIEW FOR SQL DATABASE
```

```
01 VW-SQL VIEW OF EMPLOYEES-SQL
02 PERSONNEL_ID
02 NAME
02 CITY
END-DEFINE
*
* OPEN A TAMINO LOGICAL READ LOOP
*
TAMINO. READ VW-TAMINO BY NAME
*
* SEARCH RECORD IN SQL DATABASE AND
* INSERT A NEW RECORD IF NOT FOUND OR
* UPDATE THE EXISTING ONE WITH THE DATA
* FROM TAMINO DB
SQL. FIND(1) VW-SQL WITH PERSONNEL_ID = PERSONNEL-ID (TAMINO.)
       IF NO RECORDS FOUND
         PERSONNEL_ID := PERSONNEL-ID (TAMINO.)
         NAME         := NAME         (TAMINO.)
         CITY         := CITY         (TAMINO.)
         STORE VW-SQL
         ESCAPE BOTTOM
       END-NOREC
       PERSONNEL_ID := PERSONNEL-ID (TAMINO.)
       NAME         := NAME         (TAMINO.)
       CITY         := CITY         (TAMINO.)
       UPDATE
     END-FIND
*
END-READ
*
END TRANSACTION
*
END
```

Natural for Tamino Restrictions

There are restrictions concerning the scope of the Tamino XML Schema language that can be used for creating schemas for Natural for Tamino DDM generation:

- Only Tamino XML Schema language constructors and attributes (as mentioned in *Tamino XML Schema Constructors* in the *DDM Services* section of the *Editors* documentation) are supported by Natural for Tamino. Other constructors such as `xs:any`, `xs:anyAttribute` cannot be applied in Tamino XML Schemas if you wish to use them together with Natural for Tamino.

- The functionality of `xs:import` is not supported by Natural for Tamino. This means that external schema components must not be referenced in a Tamino XML Schema suitable for usage together with Natural. In other words, a doctype definition in a Tamino XML Schema must resolve all references within this Tamino XML Schema itself if you are planning to use it together with Natural for Tamino.

- The attribute `mixed` of the constructor `xs:complexType` is only supported with its default value `false`. Natural for Tamino does not support mixed-content document definitions (as set with the specification `mixed="true"`). Using `mixed="true"` will result in an error during DDM generation.

- The level of nested structures in a Natural for Tamino DDM is limited to 99. A new DDM level is generated whenever one of the following constructors occurs in the Tamino XML Schema:

  ```
  xs:element
  xs:attribute
  xs:choice
  xs:all
  xs:sequence
  ```

- Recursively defined structures in a Tamino XML Schema cannot be used together with Natural for Tamino.

- The Tamino XML Schema language constructor `xs:choice` is mapped on a Natural group containing all alternatives of the choice. To restrict processing to one particular choice, an appropriate view with the required choice has to be created.

- Natural for Tamino only supports "closed content validation mode". Tamino XML Schemas with "open content validation mode" cannot be used together with Natural for Tamino.

- For the Tamino XML Schema language constructors `xs:choice`, `xs:sequence` and `xs:all`, a value greater than 1 of the attribute `maxOccurs` cannot be handled in the Natural data structures. Hence a value greater than 1 will always lead to an error during DDM generation.

- Natural for Tamino can handle only Tamino objects that are defined with a Tamino XML Schema as a subset of the W3C schema. Especially Natural for Tamino does not support non-XML (`tsd:nonXML`) data or instances without a defined schema (`ino:etc`).