ACL
Cancer Research UK

Advanced Computation Laboratory,
Cancer Research UK
44 Lincoln's Inn Fields
London WC2A 3PX
Tel: 020 7269 3627 Fax: 020 7269 3186

CREDO-2004-006

Generic Data Access for Tallis

Tony Rose

Creation Date	12 May 2004
Last Modification	27 May 2004
Revision	61
Version	e.g. 1
Circulation	e.g. unrestricted
Status	e.g. unreviewed

1. Introduction

This document describes a generic approach to data access from Tallis. It builds on the framework described in CREDO-2003-021 ("A Generic Data Access Framework for Tallis") but extends it in a number of critical ways:

1. The framework now supports enactment through both the Tallis Tester and through the Web Components (browser interface);

2. SQL prepared statements are now used in place of SQL statements;

3. The syntax used to map between Tallis dataitems and SQL has been greatly simplified (partially as result of the above);

4. Instead of actionHandlers and enquiryHandlers we now have four distinct handler types:

    - enquiryReaders
    - enquiryWriters
    - decisionWriters
    - actionWriters

This list of handlers is by no means exclusive: if a need for a further type of handler is identified, it can be implemented. But at present the above four cover the vast majority of situations encountered thus far. Also, it is hard to imagine a practical use for some of the other combinations, e.g. when would anyone ever need a decisionReader? What function could an actionReader perform that couldn't be better expressed by an enquiryReader?

Nonetheless, the set is inherently extensible, and if a requirement is identified then others can be implemented. Moreover, the basic functionality expressed by each can also be extended: at present their behaviour is very database-centric – i.e. they assume that data access is to and from a database. This is likely to evolve, as other types of data access become relevant, e.g. to online medical calculators, or to other knowledge bases, software components, etc.

To get maximum value from this document, the reader may wish to familiarise themselves with CREDO-2003-021, as that covers much of the basic concepts and background material which will not be repeated here.

1.1 Pre-requisites

In order to use the framework described in this document, you will need the following:

1. A version of Tallis that supports DB access. See the Tallis download site for details.

2. Some knowledge of XML. You will need to understand the XML schema described in Section 2, and be able to create an XML instance document that conforms to this schema.

3. Some knowledge of SQL. In order to interact with a database, you will need to compose statements that can be interpreted by that database. In most cases this will be SQL.

2. The General Solution

The basic idea behind the solution is to use XML files to describe the mapping between proforma dataitems and the external resources they represent in a given local context. This is best illustrated by an example. In our case, we have chosen as our application a simple, informal process for assessing whether someone has a cold or flu. This process description does not contain any serious clinical content, but it does exercise the four handler types outlined above.

The process description content (i.e. the proforma source file), along with all the other associated resources (i.e. the schema definition and the XML instance document) are listed in the appendices.

2.1 The XML Schema

The first thing the user has to do is to create an XML instance document that represents the mapping between the proforma dataitems in the process description and the SQL statements that will perform the necessary database interaction (see Appendix B). This instance document must conform to the XML schema (see Appendix C).

2.2 The Database

For tutorial purposes, we have built a simple MS Access database to support this process description. This database consists of two tables: 'PatientData', which contains some simple patient information (patient ID, patient's name, symptoms, etc.) and 'Decisions', which contains the results of the clinical decisions made when running the process description.

3. A Worked Example

3.1 The XML Instance Document

As mentioned above, the first step is to produce an XML instance document that describes the mapping between our proforma dataitems and the fields in our database. This involves the specification of values for the following schema elements:

1. the database (i.e. the parameters needed to connect to the database)

2. the task handlers (i.e. the set of enquiryReaders, enquiryWriters, decisionWriters, actionWriters relevant to the process description)

3.1.1 The root element

The root element of the XML mapping file has no declared attributes but it must possess the attributes necessary to bind it to an appropriate schema (i.e. the namespace and schema location). In our example this element is populated as follows:

```
<proformaDataMapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" xsi:noNamespaceSchemaLocation="..\proformaDataMapping.xsd">
```

3.1.2 The database element

The database element requires 2 attributes to be specified: `url` and `driver`. The remaining 2 attributes (`username` and `password`) are optional.

```
<xs:element name="database" maxOccurs="unbounded">
   <xs:complexType>
      <xs:simpleContent>
         <xs:extension base="xs:string">
            <xs:attribute name="url" type="xs:string" use="required"/>
            <xs:attribute name="driver" type="xs:string" use="required"/>
            <xs:attribute name="username" type="xs:string" use="optional"/>
            <xs:attribute name="password" type="xs:string" use="optional"/>
         </xs:extension>
      </xs:simpleContent>
   </xs:complexType>
</xs:element>
```

In our worked example, this element is populated as follows:

```
<database url="jdbc:odbc:ColdOrFluDSN" driver="sun.jdbc.odbc.JdbcOdbcDriver">
   ColdOrFluDB
</database>
```

Note that we can have more than one database element – this is so that the process description can refer to more than one database simultaneously if necessary. The name of the database ("`ColdOrFluDB`" in the above example) is then used in each SQL statement (see below) to refer to the appropriate database.

3.1.3 The Prepared SQL Statement type

The `preparedSQLStatementType` is a complex type that represents the content of a prepared SQL statement. It has two required attributes: `dbName`, which is the name of the database to which it applies, and `params`, which are used to populate the prepared statement:

```
<xs:complexType name="preparedSQLStatementType">
   <xs:annotation>
      <xs:documentation>An SQL prepared statement</xs:documentation>
   </xs:annotation>
   <xs:simpleContent>
      <xs:extension base="xs:string">
         <xs:attribute name="dbName" type="xs:string" use="required"/>
         <xs:attribute name="params" type="xs:string" use="required"/>
      </xs:extension>
   </xs:simpleContent>
```

```
</xs:complexType>
```

An example of an instantiated prepared SQL statement is as follows:

```
<preparedSQLStatement dbName="ColdOrFluDB" params="patientID patientsName
symptoms">
   INSERT INTO PATIENT_DATA (PATIENT_ID, NAME, SYMPTOMS) VALUES (?, ?, ?)
</preparedSQLStatement>
```

When this statement is executed by the engine, the dataitems named in each of the fields in `params` will be substituted with their respective values at the time of execution. Note that in this particular example the SQL table columns (i.e. `PATIENT_ID, NAME, SYMPTOMS`) have names that are similar to the dataitems themselves, but we have chosen to use upper case to differentiate them from the dataitems. Normally it would be prudent to use identical names for both (to reduce the possibility of typing errors).

3.1.4 The enquiryReader element

The enquiryReader element has one required attribute: `dataItem`, which specifies the proforma dataitem that this enquiryReader is designed to populate. Optionally, we may also provide a `taskName`, which specifies that this enquiryReader will only be invoked when the `dataItem` is being requested by a given task (typically an associated enquiry). The enquiryReader can contain an unbounded number of prepared SQL statements, which will be executed in order by the engine.

```
<xs:element name="enquiryReader" minOccurs="0">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="preparedSQLStatement" type="preparedSQLStatementType"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="dataItem" type="xs:string" use="required"/>
      <xs:attribute name="taskName" type="xs:string" use="optional"/>
   </xs:complexType>
</xs:element>
```

The result of each SQL query should contain only one column. If the data item is of type `text`, `integer`, `real`, or `Boolean`, then this column should contain only one row whose value is supplied to the data item. If the data item is of type `setof_text`, `setof_integer` or `setof_real` then the result should still contain only one column but may contain more than one row. The resulting set of values is then assigned to the data item. In our worked example the enquiryReader is populated as follows:

```
<enquiryReader dataItem="patientID">
   <preparedSQLStatement dbName="ColdOrFluDB" params="">
      SELECT (Max(PATIENT_ID)+1) FROM PATIENT_DATA
   </preparedSQLStatement>
</enquiryReader>
```

This means that when the dataitem `patientID` is requested (by any task), the SQL statement will be executed. In this case, the SQL simply returns an integer that is 1 greater than the last `PATIENT_ID` from the table `PATIENT_DATA`. This value is then assigned to `patientID` by the Tallis engine.

3.1.5 The enquiryWriter element

The enquiryWriter element has one required attribute: `taskName`, which specifies the proforma task that this enquiryWriter is designed to handle. The enquiryWriter can contain an unbounded number of prepared SQL statements, which will be executed in order by the engine.

```
<xs:element name="enquiryWriter" minOccurs="0">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="preparedSQLStatement" type="preparedSQLStatementType"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="taskName" type="xs:string" use="required"/>
   </xs:complexType>
</xs:element>
```

In our worked example this is populated as follows:

```
<enquiryWriter taskName="get_symptoms">
   <preparedSQLStatement dbName="ColdOrFluDB" params="patientID patientsName
symptoms">
      INSERT INTO PATIENT_DATA (PATIENT_ID, NAME, SYMPTOMS) VALUES (?, ?, ?)
   </preparedSQLStatement>
</enquiryWriter>
```

This means that when the action `get_symptoms` enters the completed state (i.e. all of its mandatory sources have been supplied with values), the SQL statement will be executed. In this case the `patientID`, `patientsName`, and `symptoms` will be written to the `PATIENT_DATA` table. Note that some of the values required by an enquiry could actually have been populated by enquiryReader (as is the case of `patientID` in our example).

3.1.6 The actionWriter element

The actionWriter element is similar to the enquiryWriter element, in that it has one required attribute: `taskName`, which specifies the proforma task that this actionWriter is designed to handle. The actionWriter can contain an unbounded number of prepared SQL statements, which will be executed in order by the engine.

```
<xs:element name="actionWriter" minOccurs="0">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="preparedSQLStatement" type="preparedSQLStatementType"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="taskName" type="xs:string" use="required"/>
   </xs:complexType>
</xs:element>
```

In our worked example this is populated as follows:

```
<actionWriter taskName="prescribe_painkillers">
   <preparedSQLStatement dbName="ColdOrFluDB" params="patientID">
      UPDATE PATIENT_DATA SET PILL_TAKER = TRUE WHERE PATIENT_ID = (?)
   </preparedSQLStatement>
</actionWriter>
```

This means that when the action `prescribe_painkillers` enters the state `in_progress` (i.e. its scheduling constraints, preconditions etc. have been met), the SQL statement will be executed.

Note that actionWriters are designed to 'fire' when the task state becomes *in progress* (rather than *completed*). This is because actionHandlers (and, indirectly, enquiryReaders) are based on the principle that if a task can be performed without requiring input from the user, then it should *not* be necessary to ask the user for manual confirmation. To achieve this they must have the ability to 'confirm themselves', if (and only if) at least one of their SQL statements(s) succeeds. In our example, this means that when the `prescribe_painkillers` task becomes `in_progress`, if the `PATIENT_DATA` table is successfully UPDATED, then the actionWriter will confirm the task `prescribe_painkillers` itself and the task will thus become `completed`.

3.1.7 The decisionWriter element

The decisionWriter element is similar to the enquiryWriter element, in that it has one required attribute: `taskName`, which specifies the proforma task that this decisionWriter is designed to handle. The decisionWriter can contain an unbounded number of prepared SQL statements, which will be executed in order by the engine.

```
<xs:element name="decisionWriter" minOccurs="0">
   <xs:complexType>
      <xs:sequence>
         <xs:element name="preparedSQLStatement" type="preparedSQLStatementType"
maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="taskName" type="xs:string" use="required"/>
   </xs:complexType>
</xs:element>
```

In our worked example this is populated as follows:

```
<decisionWriter taskName="diagnosis">
   <preparedSQLStatement dbName="ColdOrFluDB" params="patientID DECISION_NAME
TIME_STAMP RECOMMENDATIONS SELECTION">
      INSERT INTO DECISIONS (PATIENT_ID, DECISION_NAME, TIME_STAMP,
RECOMMENDATIONS, SELECTION) VALUES (?, ?, ?, ?, ?)
   </preparedSQLStatement>
</decisionWriter>
```

This means that when the task `diagnosis` enters the completed state (i.e. it has been confirmed and any mandatory sources have been supplied with values), the SQL statement will be executed. Note that in this case the `params` list contains a number of keywords: `DECISION_NAME`, `TIME_STAMP`, `RECOMMENDATIONS`, `SELECTION`. These keywords correspond to particular values that will be provided by the engine, and are defined as follows:

`DECISION_NAME` = A string representing name of the decision (i.e. the task name)

`TIME_STAMP` = A SQL timestamp representing the current date & time

`RECOMMENDATIONS` = A string representing the recommended candidate(s)

SELECTION = A string representing the selected candidate(s)

3.2 Associating the process description with the XML instance document

In comparison to the earlier version of this framework (described in CREDO-2003-021), the changes required to the process description itself (i.e. the .pf file) are minimal - all we need is some way of associating the .pf file with the .xml file.

For enactment via the web interface, the user just needs to add a metadata entry to the context field of the root plan. This metadata entry should be of the form: #METADATA DB.MappingFile=<URL>. In our example, this is populated as follows:

```
context :: '#METADATA
DB.MappingFile=file:///C:/Documents%20and%20Settings/tr/jbproject/TallisWor
kingDirectory/TestData/TestXMLMapper/TestWebDB/coldOrFlu.xml' ;
```

However, for enactment via the Tester, if the context field is empty then the current directory of the local filesystem will be searched for a file of the form <same_name>.xml, i.e. the Proforma file C:\TestWebDB\coldOrFlu.pf will by default be associated with the XML file C:\TestWebDB\coldOrFlu.xml. This allows users to use different XML instance documents without changing the .pf file, and hence maintains the principle that the .xml file can depend on the .pf file but not vice versa.

Evidently, it would be useful to apply this principle also to web enactment, but in distributed applications the concept of "the current directory of the local filesystem" is somewhat ambiguous (i.e. local to where – the client, the server, or perhaps some other location?). Consequently, for web enactment, the URL for the XML instance document must be specified using the context field of the root plan in the .pf file.

4. Summary

This document has presented a basic framework for database access within Tallis. It represents a significant advance on the initial version (described in CREDO-2003-021), in particular:

1. The framework now supports enactment through both the Tallis Tester and through the Web Components (browser interface);

2. SQL prepared statements are now used in place of SQL statements;

3. The syntax used to map between Tallis dataitems and SQL has been greatly simplified (partially as result of the above);

4. Instead of actionHandlers and enquiryHandlers we now have four distinct handler types:

    - enquiryReaders

- enquiryWriters
- decisionWriters
- actionWriters

The natural continuation of this work is to extend the current framework to support interaction with other applications and resources, such as online medical calculators (e.g. MEDAL), external knowledge bases, software components, etc.

Appendix A: The PROforma File

```
/** PROforma (plain text) version 1.3.37.2 **/
plan  :: 'coldOrFlu' ;
     caption ::"coldOrFlu";
     context :: '#METADATA

DB.MappingFile=file:///C:/Documents%20and%20Settings/tr/jbproject/TallisWorkingDi
rectory/TestData/TestXMLMapper/TestWebDB/coldOrFlu.xml' ;
     component :: 'get_symptoms' ;
          number_of_cycles ::1;
          ltwh :: 147,18,42,36;
     component :: 'diagnosis' ;
          schedule_constraint :: completed('get_symptoms') ;
          number_of_cycles ::1;
          ltwh :: 342,18,36,36;
     component :: 'prescribe_painkillers' ;
          schedule_constraint :: completed('diagnosis') ;
          number_of_cycles ::1;
          ltwh :: 518,18,36,36;
     component :: 'discharge_patient' ;
          schedule_constraint :: completed('diagnosis') ;
          number_of_cycles ::1;
          ltwh :: 518,18,36,36;
end plan.

action  :: 'discharge_patient' ;
     caption ::"discharge patient";
     precondition ::result_of( diagnosis )  = COLD  OR result_of( diagnosis)  =
WORKITIS;
     procedure ::patientsName # " has been discharged.";
end action.

decision  :: 'diagnosis' ;
     caption ::"diagnosis";
     candidate :: 'WORKITIS' ;
          recommendation ::netsupport(diagnosis, WORKITIS) >= 1;
     candidate :: 'COLD' ;
          argument :: for,symptoms  includes sore_throat      attributes
               argument_name :: 'symptoms  includes sore_throat     ' ;
          end attributes
          ;
          recommendation ::netsupport(diagnosis, COLD) >= 1 AND
netsupport(diagnosis, FLU) < 1 ;
     candidate :: 'FLU' ;
          argument :: for,symptoms  includes high_temperature  OR symptoms
includes shivers      attributes
               argument_name :: 'symptoms  includes high_temperature  OR symptoms
includes shivers     ' ;
          end attributes
          ;
          recommendation ::netsupport(diagnosis, FLU) >= 1;
end decision.

action  :: 'prescribe_painkillers' ;
     caption ::"prescribe painkillers";
     precondition ::result_of( diagnosis )  = FLU;
     procedure ::patientsName # " has been added to the list of pill poppers.";
end action.

enquiry  :: 'get_symptoms' ;
     caption ::"get symptoms";
     source :: 'patientID' ;
          mandatory :: no ;
```

```
     source :: 'patientsName' ;
     source :: 'symptoms' ;
end enquiry.

data  :: 'patientID' ;
     type :: integer ;
end data.

data  :: 'patientsName' ;
     type :: text ;
     caption ::"What is the patient's name?";
end data.

data  :: 'symptoms' ;
     type :: setof_text ;
     caption ::"What symptoms does the patient have?";
     range ::"high_temperature","shivers","sore_throat","nothing";
end data.
```

Appendix B: The XML Instance Document

```
<?xml version="1.0" encoding="UTF-8"?>
<!--Sample XML file generated by XMLSPY v5 rel. 3 U (http://www.xmlspy.com)-->
<proformaDataMapping xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="..\proformaDataMapping.xsd">
   <database url="jdbc:odbc:ColdOrFluDSN"
driver="sun.jdbc.odbc.JdbcOdbcDriver">ColdOrFluDB</database>
   <enquiryReader dataItem="patientID">
      <preparedSQLStatement dbName="ColdOrFluDB" params="">
         SELECT (Max(PATIENT_ID)+1) FROM PATIENT_DATA
      </preparedSQLStatement>
   </enquiryReader>
   <enquiryWriter taskName="get_symptoms">
      <preparedSQLStatement dbName="ColdOrFluDB" params="patientID patientsName
symptoms">
         INSERT INTO PATIENT_DATA (PATIENT_ID, NAME, SYMPTOMS) VALUES (?, ?, ?)
      </preparedSQLStatement>
   </enquiryWriter>
   <decisionWriter taskName="diagnosis">
      <preparedSQLStatement dbName="ColdOrFluDB" params="patientID DECISION_NAME
TIME_STAMP RECOMMENDATIONS SELECTION">
         INSERT INTO DECISIONS (PATIENT_ID, DECISION_NAME, TIME_STAMP,
RECOMMENDATIONS, SELECTION) VALUES (?, ?, ?, ?, ?)
      </preparedSQLStatement>
   </decisionWriter>
   <actionWriter taskName="prescribe_painkillers">
      <preparedSQLStatement dbName="ColdOrFluDB" params="patientID">
         UPDATE PATIENT_DATA SET PILL_TAKER = TRUE WHERE PATIENT_ID = (?)
      </preparedSQLStatement>
   </actionWriter>
</proformaDataMapping>
```

Appendix C: The XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by Tgr (Cancer
Research UK) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
   <xs:element name="proformaDataMapping">
      <xs:annotation>
         <xs:documentation>Specifies a mapping between proforma data items and DB
values</xs:documentation>
      </xs:annotation>
      <xs:complexType>
         <xs:sequence maxOccurs="unbounded">
            <xs:element name="database" maxOccurs="unbounded">
               <xs:complexType>
                  <xs:simpleContent>
                     <xs:extension base="xs:string">
                        <xs:attribute name="url" type="xs:string"
use="required"/>
                        <xs:attribute name="driver" type="xs:string"
use="required"/>
                        <xs:attribute name="username" type="xs:string"
use="optional"/>
                        <xs:attribute name="password" type="xs:string"
use="optional"/>
                     </xs:extension>
                  </xs:simpleContent>
               </xs:complexType>
            </xs:element>
            <xs:choice maxOccurs="unbounded">
               <xs:element name="enquiryReader" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="preparedSQLStatement"
type="preparedSQLStatementType" maxOccurs="unbounded"/>
                     </xs:sequence>
                     <xs:attribute name="dataItem" type="xs:string"
use="required"/>
                     <xs:attribute name="taskName" type="xs:string"
use="optional"/>
                  </xs:complexType>
               </xs:element>
               <xs:element name="enquiryWriter" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="preparedSQLStatement"
type="preparedSQLStatementType" maxOccurs="unbounded"/>
                     </xs:sequence>
                     <xs:attribute name="taskName" type="xs:string"
use="required"/>
                  </xs:complexType>
               </xs:element>
               <xs:element name="actionWriter" minOccurs="0">
                  <xs:complexType>
                     <xs:sequence>
                        <xs:element name="preparedSQLStatement"
type="preparedSQLStatementType" maxOccurs="unbounded"/>
                     </xs:sequence>
                     <xs:attribute name="taskName" type="xs:string"
use="required"/>
                  </xs:complexType>
               </xs:element>
               <xs:element name="decisionWriter" minOccurs="0">
                  <xs:complexType>
```

```
                    <xs:sequence>
                       <xs:element name="preparedSQLStatement"
type="preparedSQLStatementType" maxOccurs="unbounded"/>
                    </xs:sequence>
                    <xs:attribute name="taskName" type="xs:string"
use="required"/>
                 </xs:complexType>
              </xs:element>
           </xs:choice>
        </xs:sequence>
     </xs:complexType>
  </xs:element>
  <xs:complexType name="SQLStatementType">
     <xs:annotation>
        <xs:documentation> A simple SQL statement</xs:documentation>
     </xs:annotation>
     <xs:simpleContent>
        <xs:extension base="xs:string">
           <xs:attribute name="dbName" type="xs:string" use="required"/>
        </xs:extension>
     </xs:simpleContent>
  </xs:complexType>
  <xs:complexType name="preparedSQLStatementType">
     <xs:annotation>
        <xs:documentation> An SQL prepared statement</xs:documentation>
     </xs:annotation>
     <xs:simpleContent>
        <xs:extension base="xs:string">
           <xs:attribute name="dbName" type="xs:string" use="required"/>
           <xs:attribute name="params" type="xs:string" use="required"/>
        </xs:extension>
     </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```