ENABLING THE FUTURE

Natural / Adabas

Migration Solutions

RMEn - Nat2EGL Process Guide

October, 2014

Table of Contents

1   Objective

This document is intended to provide an understanding of the step-by-step process for migration of a Natural/ADABAS application to a functionally equivalent EGL/DB2 or EGL/Oracle based application using the IBM Rational Software Development Platform and FBD Associates Inc. (FBDA) RMEn-Nat2EGL migration tools. These steps are illustrated in Figure 1 and from the beginning to the completion of the process are:

- Collection of Natural source code and Adabas FDTs

- Natural source code parsing

- Loading the parsed data to a repository

- Analysis and assessment

- Application partitioning

- Adabas FDT normalization and DB2/Oracle relational schema generation

- Conversion of Natural source code to EGL

- Adabas data conversion and migration

- Testing and user acceptance
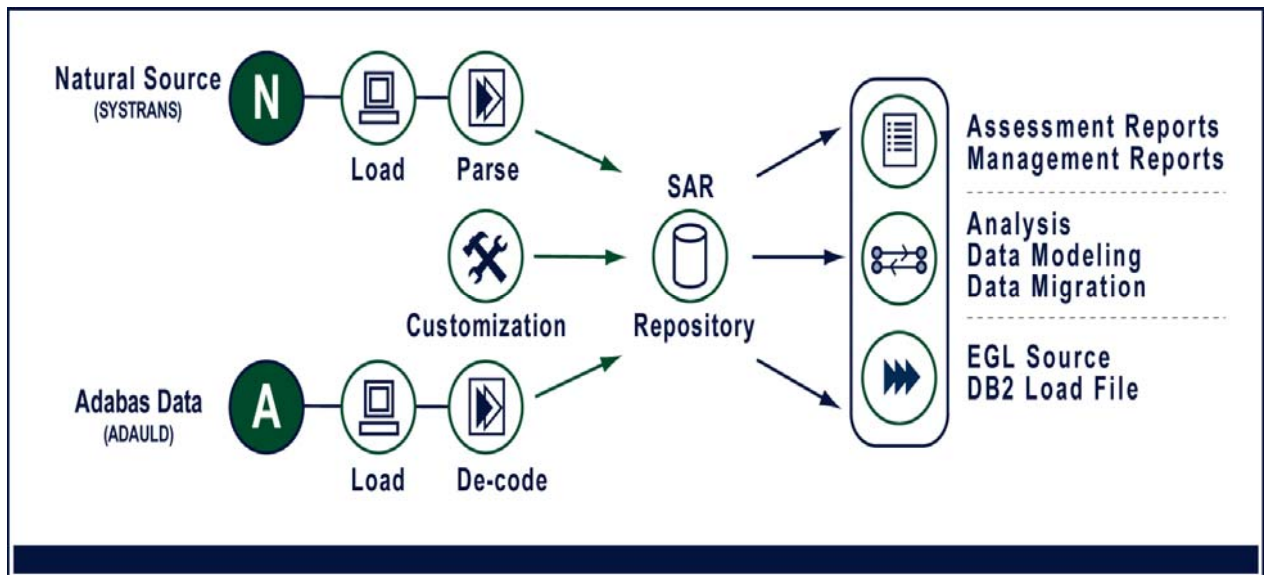
Figure 1   RMEn-Nat2EGL Process

2   Collection of Customer Source Code

The process begins with the collection of all the Natural source code from the customer's application portfolio in SYSTRANS format. The SYSTRANS must include the Adabas Field Definition Tables (FDTs) as well as the Natural Data Definition Modules (DDMs), Maps, Data Definition Areas (global, local and parameter data areas), and executable modules (programs, subprograms, subroutines, helproutines). This can be accomplished using the Software AG ADAREP and SYSTRANS utilities.

To assist in this activity, IBM Rational and FBDA will provide special instructions for source code collection. For details regarding the collection of source code contact IBM Rational.

The SYSTRANS files can be emailed or ftp'd to IBM Rational/FBDA for parsing. Where STEPLIBS are used, the STEPLIB information must be provided.

3   SYSTRANS Analysis

Written in Java and using Apache Derby (an open source relational database), the Systrans Auditor reads the SYSTRANS files provided by a customer and produces a series of reports showing basic application statistics (module and line counts) for the application as well as reports that:

a)   identify missing Natural modules and indicate where they are called from

b)   suggest STEPLIBS that may serve to eliminate some of the missing Natural modules

c)   identify certain modules which, though missing, can be ignored (for example, modules that are confirmed by the customer to be irrelevant, Construct support modules, or SAG USR* modules)

d)   identify 3GL calls and the modules which call them

The use of the SYSTRANS Auditor assists with the missing modules analysis and ensures that the code base is complete prior to the more labor and processing intensive parsing and analysis activities.

The document "Instructions for SYSTRANS Auditor" provides additional technical and operational details about the SYSTRANS Auditor and is available upon request.

4   Parsing and Analysis

The FDTs, DDMs, Maps and other Natural objects from the SYSTRANS are parsed, and the parsed data extracted and loaded into the relational based Syntax Analysis Repository (SAR). Initially, an automated analysis of the parsed data is performed using SNAP® to determine the feasibility of conversion and identify problematic code (e.g.: dynamic code, external calls). Detailed application profile assessment reports (see below) are generated, containing a comprehensive description of the source code and application details which aid in estimating the size of the migration.

SNAP® is an interactive web-based tool that reads the SAR database and allows the user to perform a number of functions to support application conversion and data migration.
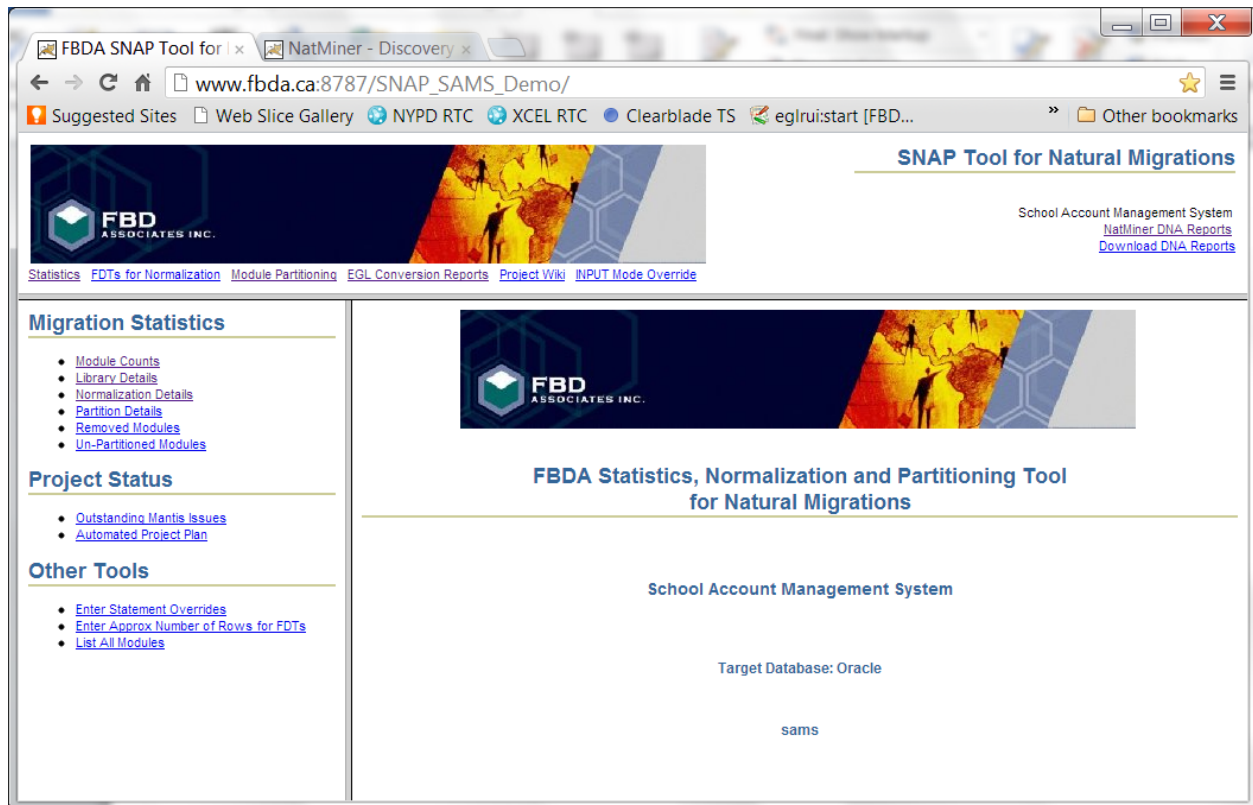
Using SNAP® one can:

- View statistics

- Normalize Adabas FDTs based on DDMs

- Create the DDL for schema definition, the control file and SQL for database triggers

- Read Adabas data files (ADAULD) and produce files for relational database loading

- Manage partitioning of applications (and libraries) in preparation for conversion

- View EGL conversion reports post-conversion to assess the remediation effort

- View and/or download Natminer® DNA Reports

NatMiner® DNA Reports provides comprehensive inventory reports that ensure the completeness and integrity of the code prior to conversion. During the assessment, any modules and/or DDMs that are identified as unused will be purged from SAR database before proceeding with subsequent processing stages.

NatMiner - Discovery aNd Analysis (DNA) Reports - sams

NatMiner DNA Profile Home   FBDA Home

NatMiner - Discovery aNd Analysis (DNA) Reports

Inventory Reports
- Metrics
- Migration Metrics

Audit Reports
- Modules Not Parsed
- Missing Modules
- Missing DataAreas
- Missing DDMs
- Missing FDTs
- Missing Copycodes
- Unresolved Dynamic Calls
- Duplicate Module Names
- Subroutines Not Called

Analysis Reports

Data Areas
- Not Referenced

DDMs
- Referenced
- Not Referenced
- Missing FDT

FDTs
- Referenced
- Not Referenced

Usage Reports
- Data Areas
- Maps
- Helps
- DB Access
- FDTs
- Work Files
- 3GL Calls
- Super/Sub Descriptor Report
- STEPLIBs Report

Conversion Reports
- Reinput Statements
- Retry Statements
- Escape Top/Bottom Issues
- Construct Generated Modules
- SAG SYSEXT USR... Modules
- Maps with Proc Rules
- Input No Erase
- Helps

FDTs/DDMs
- DDMs
- FDTs

Natural Libraries
- SAMS

Key Data

Date Generated	Tuesday, October 16, 2012 10:57 AM
Total Modules	135
Total Lines	13,560
Average Lines of Code per Module	100
Number Of Libraries	1
Steplibs Exist	No
Number Of Modules Not Parsed	0
Number Of Reinputs	86
Number Of Modules Affected By Reinputs	21
Number Of Input No Erases	0
Number Of Modules Affected Input No Erases	0

Using Natminer® you can:

- View Application Profile Reports including general information such as:

  - application metrics
  - modules not parsed
  - missing objects (executables, ddm, data areas, copy code)
  - un-referenced subroutines
  - dynamic calls

- View reports showing potential conversion issues

- View source code in HTML format

SNAP® and NatMiner® provide a convenient way to view the parsed and converted applications, perform the database normalization, and a good basis for planning all phases of the project.

5   Partitioning

Using SNAP® together with the Natural source code and any available documentation, the application is analyzed and organized into logical units of work (partitions) that will be tested and delivered incrementally consistent with customer test and production support resource availability. Partitions will be determined for the most part by functions within the application. Several functions that are made up of only a few modules may be combined into one partition while other, more complex functions may be split over more than one partition.

Partitions are designed to follow the normal business procedures. For example, security and support file maintenance would be likely candidates for the first partition. Subsequent partitions would then use and build on the files already created and populated.

While it is not essential, it is highly desirable that the customer be involved in partitioning from the beginning. The proposed plan will be reviewed, amended as required and ultimately approved by the customer before work begins.

The source code translation will be implemented in phases with each phase consisting of one or more of these partitions. As the source code for each partition is translated it will be compiled and subject to unit level test to confirm functional performance.

6   Normalization

6.1   Automated Normalization

The Parsing and Analysis phase of the NAT2EGL process ensures completeness of the client's code. The next phase is normalization of the FDTs to create a relational schema. The generation of the DDL is done together with the source conversion. The DB2/Oracle database schema is derived from the ADABAS File Definition Tables (FDTs) using the ADABAS FDT Normalization tools. Customers provide ADABAS data as an ADAULD (ADA Unload) file. ADAULD is a standard utility available to ADABAS users for unloading all data for a given FDT.

The relational tables (DB2/Oracle) will be built from the conversion details stored in SAR, according to the following mapping rules:

- Each FDT is converted to at least one corresponding relational table in the target DBMS whose unique key will be the ISN of the original Adabas record. This table will be comprised of all the fields from the FDT with the exception of Periodic Groups (PEs), Multiple Usage fields (MUs) and Group fields.

- PEs are normalized, becoming Child tables with Foreign Keys to the parent table.

- Similarly, MUs are normalized, and Child Tables created with Foreign Keys to the Parent Table. Optionally, an MU field may be mapped to a partitioned text field that contains all occurrences of the MU delimited by a special character, or a separate field may be created to hold each occurrence of the MU field. (e.g.: ADDRESS_LINE1, ADDRESS_LINE2, etc.)

- Each Superdescriptor (SP) and Subdescriptor (SB) is converted to a physical field in the corresponding table in the target database. A database trigger is created which maintains the structure of the SP or SB components when these are modified in the database.

- Descriptors become indexes in the new schema.

- Phonetic descriptors will be replaced by database triggers that implement a Soundex algorithm.

Schema generation is an iterative process involving customer review and input until a satisfactory schema is agreed upon.

At this point, DDL is generated, including:

- CREATE TABLE, PRIMARY KEY and FOREIGN KEY statements

- CREATE INDEX statements

- Stored procedures to manage sub/superdescriptor triggers

- ISNs are created as identity columns

Additionally, the migration tool generates:

- Relational Mapping files (RMF) needed for the data extraction process

- DB2/Oracle Load commands that correspond to the new relational schema

6.2   Client Input to Conversion Relational Files

The customer has the option of specifying field and table names and some ability to change data types and sizes that will be used in the creation of the new schema. While changes may be provided in a number of formats (e.g. Excel, CSV, etc) they are entered into SAR using SNAP®.

The Nat2EGL default field and table names for FDTs are cryptic, as they are built from the Adabas short names. However, the customer has the option of overwriting table and field names so that the new schema definition will be more meaningful to application developers.

When normalizing an FDT, the DDM fields are presented side by side with the FDT information in SNAP®, and can be used as a default for the new name. In the example below, there is only one DDM for the FDT.
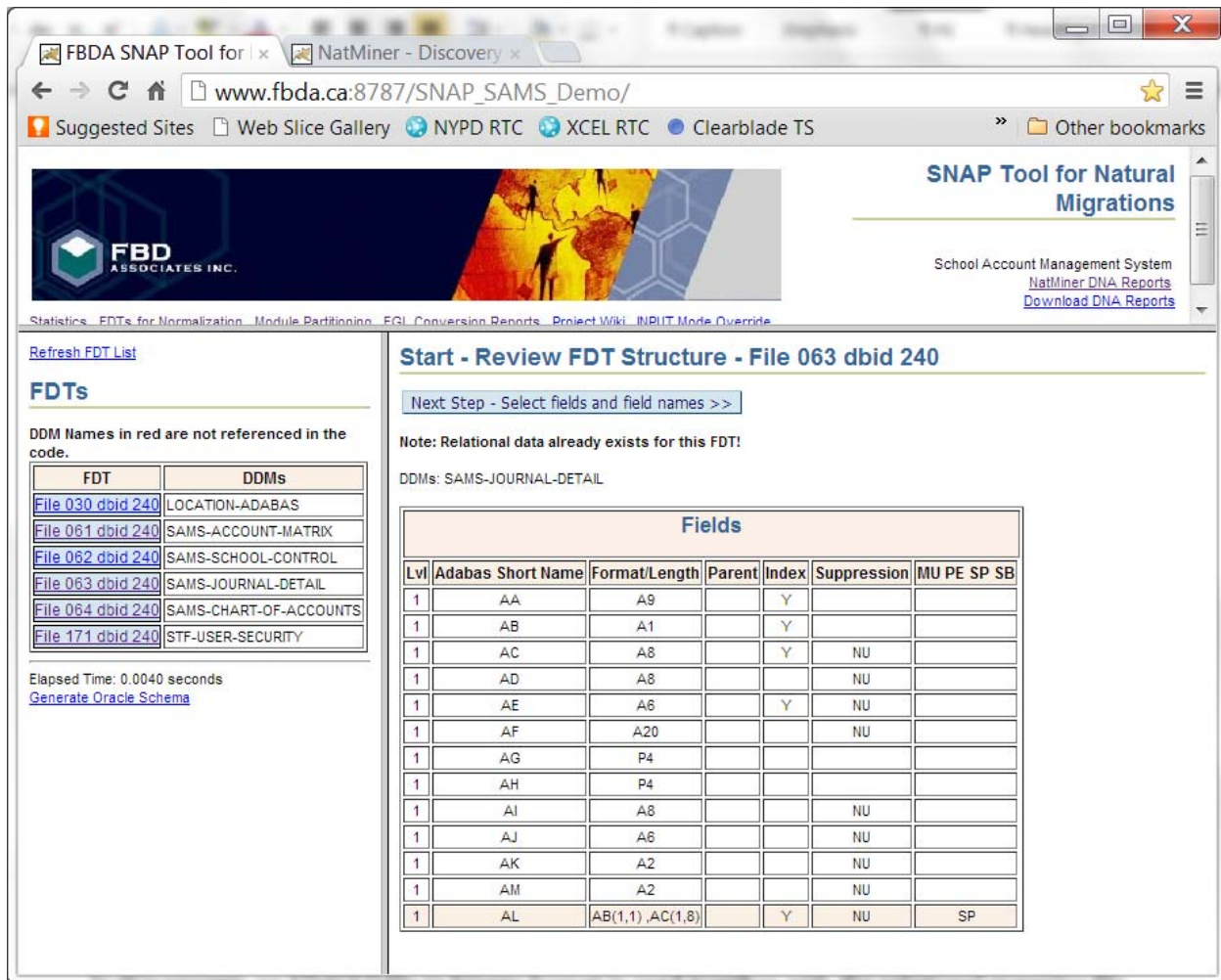
A person with a passing knowledge of Natural/ADABAS would have no difficulty in interpreting the above fields.

When the steps have been completed and the file normalized satisfactorily, clicking on Generate Oracle Schema in the left pane will complete the process, creating among other things, a Relational Mapping file (RMF) needed for the data extraction process.

6.3   ADAULD Extraction

In this process, an ADAULD file in binary format is read together with the relational mapping file (RMF), and the ADAULD data is then mapped to the table and field definitions that were created during the normalization process. The extraction produces a flat file containing the converted data in a format which can be loaded into the new DB2/Oracle schema using the DB2/Oracle load commands created during migration process.

The extraction process also creates a set of batch files that allows for quick and easy execution of the DB2/Oracle load commands created during migration.

7   Source Code Conversion

7.1   Automated conversion to EGL

Upon completion of database normalization, the source code can be converted. To ensure the generation of the correct DML it is essential that the FDT normalization details be stored in the SAR before proceeding.

The Nat2EGL source code conversion reads the SAR database and produces EGL equivalent code. Nat2EGL comments are inserted where applicable, and are of three types:

- M (Message) is strictly for your information and requires no action

- R (Requires review) indicates that the generated code should be reviewed to ensure that it will function correctly, and appropriate remedial action taken if required

- W (Warning) indicates that some additional action will be required – generally a new statement not yet handled in the converter

Please refer to Appendix A for a side-by-side comparison of a sample Natural/ EGL program.

7.1.1   Artifacts created in conversion

Many of the artifacts produced during conversion are independent of the environment (i.e. Batch Versus Online). Others are unique to online.

For both Batch and Online, the following EGL packages are created, each in a separate directory:

DDM (Batch and Online)

For each Natural DDM an object of type 'sqlRecord' is created in EGL. The converted view objects are placed in a directory called 'ddm'. This is imported into EGL as a package that is referenced by modules that have database access.

Library (Batch and Online)

For each Natural module of type (Program, Subprogram, Subroutine, Map, Helproutine, GDA, PDA, LDA) an object is created in EGL. These objects are placed in a directory named for the Natural library of origin. Together these packages which correspond to Natural Libraries comprise the main package in EGL.

- Executable objects (programs, subprograms, subroutines, helproutines) become EGL 'called' programs.

  - For **Batch**, these have a type 'basicProgram()'

  - For **Online**, i.e. modules that have a converse, type is 'UIProgram()'

- For each Map an EGL handler is generated of type 'basicHandler'

- Global Data Areas (GDA) become EGL objects of type 'Library'.

- Other Data Definition Areas (PDA, LDA) become EGL objects of type 'basicRecord'.

Additional library packages for Maps (Online only)

In addition to the basicHandlers mentioned above, for each Map or Input (from a screen) statement the following related objects are created in 2 separate packages:

- an EGL handler of type 'RUIHandler'

- a data record of type 'basicRecord'

Nat2Egl_Support_Lib (Batch and Online)

A project called 'Nat2Egl_Support_Lib' is also supplied. This package contains EGL source objects that are created by FBDA to provide functionality similar to Natural. Some examples are the code that handles PF key assignment, support for Natural Stack commands and certain statements such as Examine. This support library must be included on the EGL build path in order for the converted code to compile.

The Nat2Egl_Support_lib is environment-independent except for a couple of libraries and functions that must be included for online (RUI) deployment and which are commented out for COBOL deployment.

7.2   Manual Remediation

A limited number of manual remediations (e.g. REINPUT statement, certain date/time manipulations) are required to the converted code to ensure a successful compilation and functioning of these statements.

The converted EGL versions of maps, ddms and data areas compile with no further changes.

8   Data Migration

Nat2EGL also provides tool-supported data migration from Adabas production files to the relational tables. An automated process that can be managed using ADABAS Normalization tool or independently, combines data from SAR with the newly created relational schema to enable the following:

- Creation of a relational based data staging area derived from the transformed Adabas schema.

- Population of the staging area with data from the Adabas files to facilitate subsequent data validation, data type mapping and data warehousing activities. The Adabas data will be extracted from data files produce using standard Adabas utilities.

- Generation of control files based on the new relational schema, that can be used to create files for bulk loading of the Adabas data into the production relational tables.

The data migration process can be repeated as many times as needed. Data migration will almost always require some degree of data validation and mapping.

Refer to Appendices C and D for further details.

Performance

All application platforms and databases have specific issues when it comes to performance and customization. Once customers have received the EGL source code, they may do in-house customization or they can contract for this service on an as required basis. Particular performance issues that may be identified during testing can be tuned in the following ways:

- DBAs can tune the new relational database using such techniques as adding or removing database indexes or possibly making changes to the new schema that requires an automated re-conversion of the application.

- Changing the DML for better performance.

9   Application Testing

The Nat2EGL conversion process is based on a statement to statement conversion approach. As a consequence there is a one-to-one correspondence between converted EGL code and the Natural application. The logic and business rules are identical to the original application, and the user interface has the same appearance unless it has been customized at the user's request.

The Nat2EGL conversion process simplifies testing in that testing can be done in parallel with the existing application using a Natural test environment. Existing test scripts can be applied without change. Database updates and reports can be compared to ensure that the converted application is functioning as required.

The screen image shown below is from an EGL application. It illustrates the similarity to a Natural 3270 screen map.

It is recommended that 2 environments be set up for testing in both the EGL and the Natural environment as follows:

9.1.1   Unit test environment

The purpose of Unit test is to ensure that each screen and module, in isolation, compiles and operates with a reasonable behaviour. Ensuring functionally sound application behaviour is not the purpose of this phase of testing. Testing in this phase is not closely controlled, but includes the unit test issues of field data extremes, PF key operation, error message generation, and no abends of code.

9.1.2   Acceptance test environment

This environment will be strictly controlled and testing will follow scripts which take the user through a logical cycle(s) of processing including both batch and online. Where 'parallel' testing is discussed below it is with this environment in mind.

See Appendix B for a sample test script.

9.2   Data Conversion

Verification of the data conversion is accomplished by way of an automated comparison between the converted relational tables and the original data.

The client will provide a flat file dump of the data prior to conversion. To avoid the possibility of interim change, this should accompany the ADAULD when it is provided for testing. A dump of the converted data in the same format would be obtained using IBM Rational/FBDA tools. These two files can then be compared using standard 'diff' software and the results reviewed.

9.3   Parallel testing

The ADABAS data supplied for testing is loaded into a Natural test environment. The converted data is loaded into relational tables. Using 2 terminals (or 2 screens on the same terminal) users perform the same functions in both systems and compare screens. For parallel testing it is essential that scripts be prepared and followed rigorously.

These tests will follow the process from beginning to end, including such things as rollover, month end, year end and any other process cycles, batch reports and any external interfaces there may be. Parallel testing will use client scripts where available, plus additional tests to exercise every part of the system.

Verification should be done through a regular comparison of:

- database contents

- printed outputs (reports)

- output (work) files produced

9.3.1   Macro level

Upon completion of testing a business cycle (e.g. day, month, transaction, or other business process unit), the data comparison will be done between the ADABAS and relational files and any other outputs produced.

9.3.2   Micro level

Note will be taken of every field that appears on a screen. For example, attention should be given to:

- date and time formats

- negative numbers

- specials editing of fields (edit masks such as 'XXX'-'XX'-'XXX')

- leading zeros

- trailing blanks

- decimal places

- upper/lower case

- functioning of all PF keys

- error messages

- position of cursor on entering screen

- position of cursor when validation error occurs

9.4   Multi-user testing

It is desirable upon completion of a first pass of unit and system testing, to subject the application to a multiple-users test. As early as possible, testing that involves two or more simultaneous users would be performed. This type of testing will be scheduled as partitions are rolled out, not deferred until the project is completed.

9.5   Problem reporting

Problem reporting and resolution is a process that will enable the client and other interested parties to be fully informed on a regular and on agreed frequency basis regarding the progress of code conversion.

Problems may be of types:

i)   different behaviour from the existing Natural application

ii)   same behaviour as the existing Natural application, but is deemed that the existing Natural application contains a problem

Each problem will be fully described and assigned a priority and severity.

Each problem of type (i) is the responsibility of IBM Rational/FBDA to respond with:

- An estimate of the timeframe to address the problem
- A resolution to the problem

Each problem of type (ii) will be addressed in association with the client to respond with:

- A resolution to the problem
- An estimate of the timeframe to address the problem
- A change request (CR) order to fix the problem

10 Appendix A: Sample of converted EGL code

Natural: PROGRAM 'SAMPLE' LIBRARY = 'FBDALIB'	EGL Equivalent
<pre>*************************************
**
** FBDA001   This program reads…
**
****
DEFINE DATA
**
** Global data area
**
GLOBAL USING FBGDA01
**
** Local data
**
LOCAL
**
1 LOC VIEW OF LOCATION
  2 LOCATION-CODE   (A6)
  2 DESCRIPTION   (A20)
**
1 #DESCRIPTION   (A20)
END-DEFINE
**   Program logic
**

PERFORM INIT-FIELDS
PERFORM GET-RECORDS
**
**   Subroutines
**
DEFINE SUBROUTINE INIT-FIELDS
MOVE 'OTT' TO FBGDA01.##LOC-CODE
END-SUBROUTINE
**
DEFINE SUBROUTINE GET-RECORDS
FIND LOC WITH LOCATION-CODE
       = FBGDA01.##LOC-CODE
  MOVE DESCRIPTION TO #DESCRIPTION
END-FIND
END-SUBROUTINE
**
END</pre> | <pre>package fbda.test.fbdalib;
import ca.fbda.Nat2EGL.*;
import com.sync.tx3.egl.srv.UIProgramGatewayLib;
import com.sync.tx3.egl.srv.GatewayContext;
import fbda.test.uirecords.*;
import fbda.test.ddm.*;
//***********************************
//
// Fbda001 This program reads...
//
//***********************************
Program sample type basicProgram ()
{ throwNrfEofExceptions = no, TextLiteralDefaultString = no }

 // Program Variables
   local Sample_ws;

 // GDA
   use Fbgda01;

 // view references
   use Location_Adabas_Library;
   location Location;

// Program Entry Point
   function main()
     mainLogic();
   end

   Function mainLogic()
     initFields();
     getRecords();
   end //mainLogic
//
//   Subroutines
//
Function initFields()
 Fbgda01.$$loc_code = "OTT";
end //initFields
//
Function getRecords ()
 open rs_location for location into
   local.location.location_Code,
   local.location.description
 with #sql
 {
   select LOCATION_CODE,DESCRIPTION
   from LOCATION
   where LOCATION_CODE =:Fbgda01.$$loc_code
 };
 // Find Loop
 while (location not noRecordFound)
    get next location from rs_location;
    if(location not noRecordFound)
      location _Var.sCounter = location _Var.sCounter + 1;
      move location to local. location byName;</pre> |

```
          else
            exit while;
          end
     local.$description = local.location.description;
   end //while (Find)
  //
end //getRecords
end //Program

// Working Storage Record
record Sample_ws type basicRecord
   10 loc; //LOCATION
      20 location_Code CHAR(6);
      20 description CHAR(20);
//
   10 $description CHAR(20);

end
```

11 Appendix B: Sample User Acceptance test script

Section 1: General Information		
Primary User's Name: John Doe		Test Case ID #:
User's Extension #: 123-4567		Tester's Name:
Origin:	☒ Payroll ❑ Other	❑ New Script ❑ Addendum to: #_____ ❑ Related to: #_____
Action to be Tested: Entry of extra time for principal.		Original Test Date: / /
Department Name:		Retest (1) Date: / /
Department ID:		Retest (2) Date: / /
Test Priority ❑ Show-stopper ❑ High ❑ Medium ❑ Low		

Section 2: Test Scenario (attach all detailed information used for input)
1. Navigation > Select Function "E", Pay Period 2, Location 072 <enter> 2. Type "PR" for Select Group <enter> 3. Type LASTNAME, FIRST NAME (of employee) in Starting Name Search. <enter> 4. Type "A" in Action, <tab> to enter Month and Day, <tab> to enter hours and minutes, <tab> to enter budget code*, <tab> to enter brief description. <enter>   *ABC200HT705

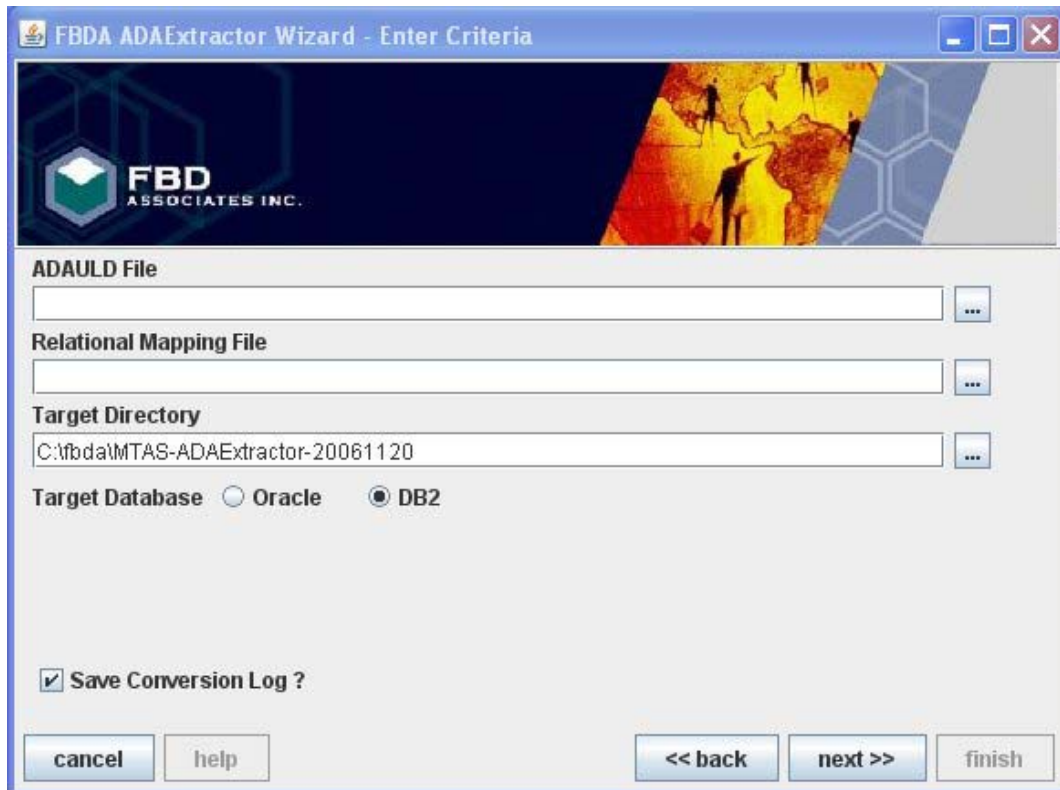| Section 3: Expected Results |
|---|
| EXPECT TO SEE "RECORD ADDED" MESSAGE DISPLAYED IN LOWER LEFT OF SCREEN. |

Section 4: Actual Results

Section 5: Test Summary		
Were original test results met?	❑ Yes ❑ No	Date PDR Submitted:
Were re-test (1) results met?	❑ Yes ❑ No	
Were re-test (2) results met?	❑ Yes ❑ No	
Tester's Signature:	Date:	
Verification of Results- Signature:	Date:	
User Acceptance/Approval:	Date:	

12 Appendix C: Relational Mapping File

Relational Mapping File (RMF)

- The relational mapping file (RMF) is created by the Natural to EGL converter at the same time as the DDL and DB2/Oracle load control files are created.

- The RMF is used by the ADAExtractor Data Migration tool to help convert the raw ADAULD file into a DB2/Oracle load file.

- The RMF contains information about the new relational fields (as defined by the Online ADABAS Normalization process) that are required by the ADAExtractor tool to create the DB2/Oracle load file correctly.

- The RMF is design to be a portable and simple way to deliver details from the SAR repository to the customer so they can do the data conversion on-site.

- The RMF, the DDL and the DB2/Oracle load control file can be generated and delivered to the customer as soon as the normalization is done. This would allow the customer to create the DB2/Oracle database and perform data conversions early in the migration process.

- Following are some screen shots of the ADAExtractor data conversion:

FBDA ADAExtractor Wizard - Enter Criteria

Select ADAULD File To Convert

Look In: MTAS-ADAExtractor-20061120

FILE_074_DB_001
FILE_094_DB_001
FILE_095_DB_001
FILE_238_DB_001
ADAExtractor.jar
charsets.jar

FBDA - ADAExtract
FILE_074_DB_001.
Load001_UYAU74.
UYAU74.dat

File Name:

Files of Type: All Files

Open   Cancel

cancel   help   << back   next >>   finish

FBDA ADAExtractor Wizard - Enter Criteria

FBD ASSOCIATES INC.

ADAULD File

C:\fbda\MTAS-ADAExtractor-20061120\FILE_074_DB_001\Datos-file74   ...

Relational Mapping File

C:\fbda\MTAS-ADAExtractor-20061120\FILE_074_DB_001\FILE_074_DB_001.rel   ...
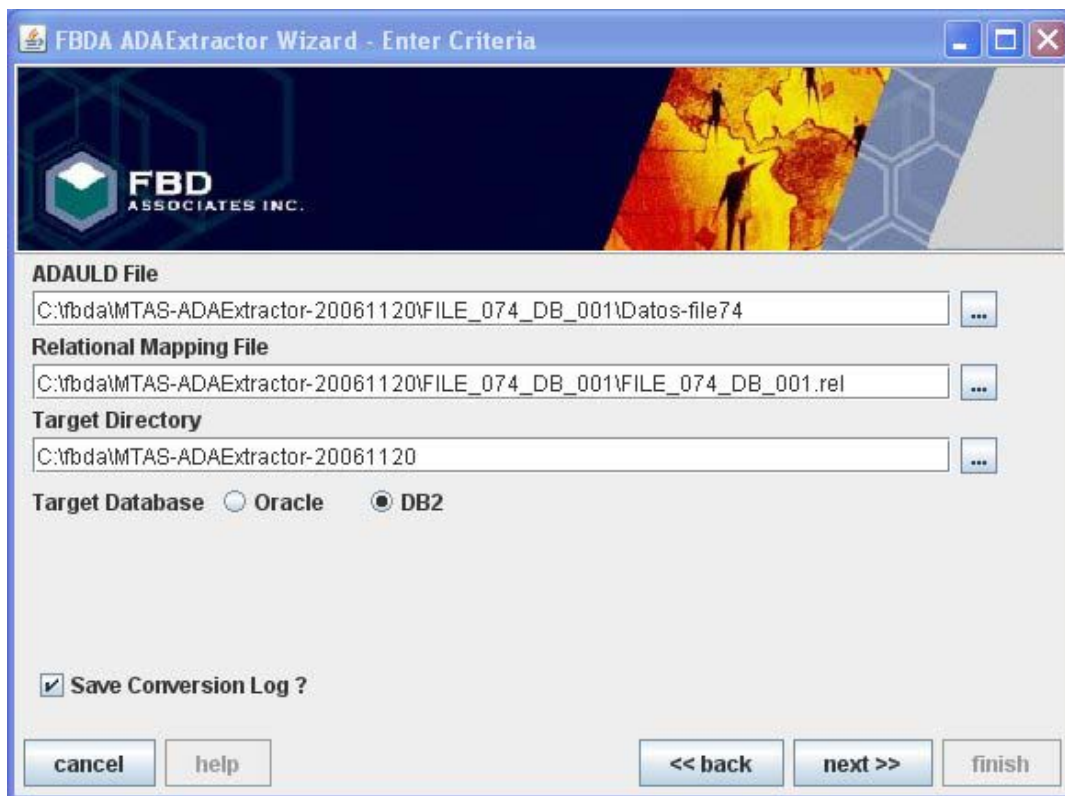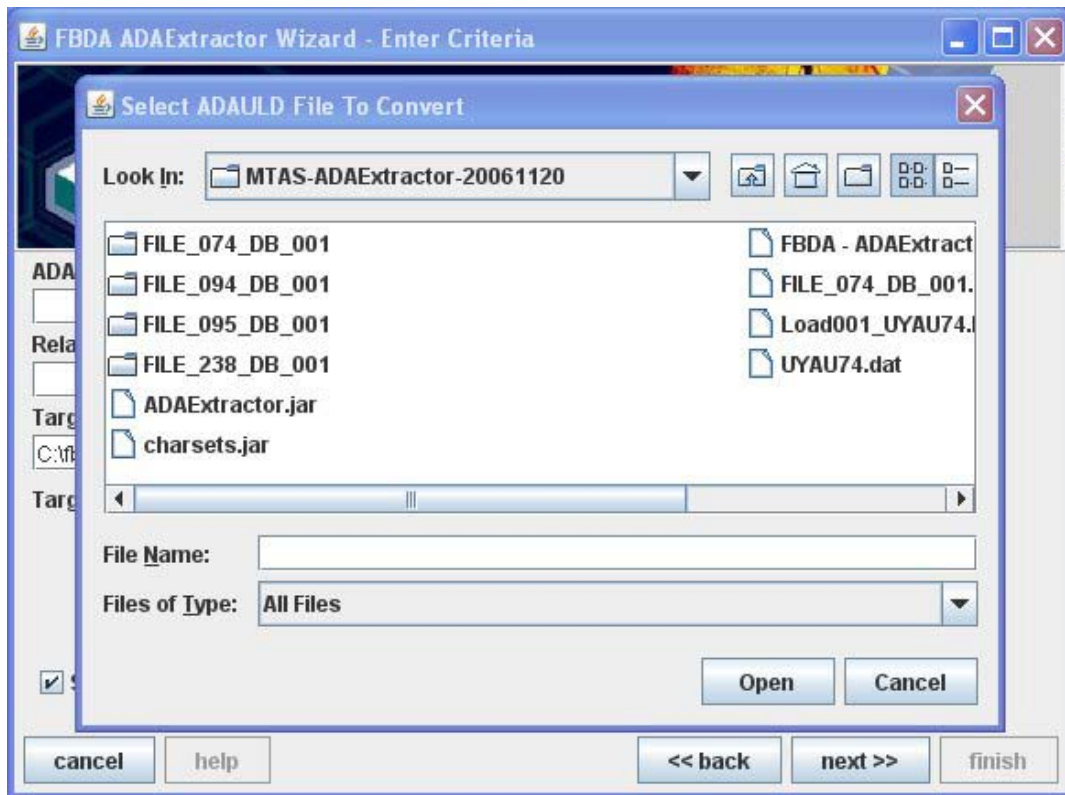
Target Directory

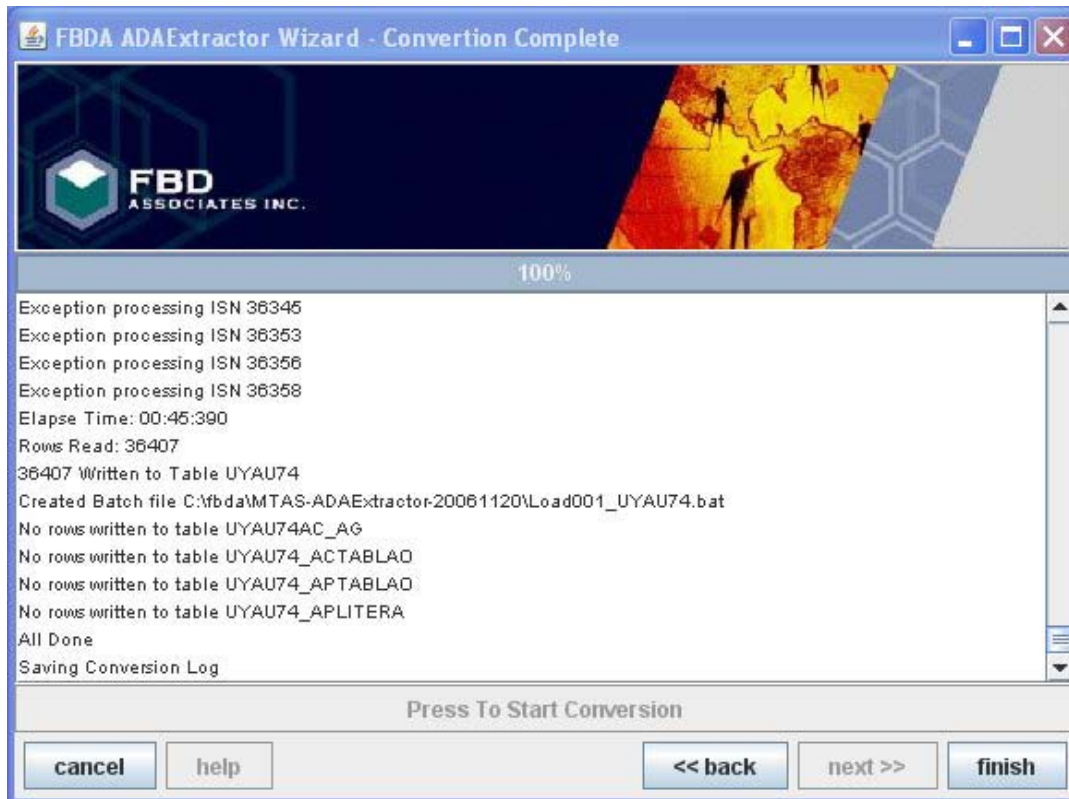C:\fbda\MTAS-ADAExtractor-20061120   ...

Target Database ○ Oracle   ● DB2

☑ **Save Conversion Log ?**

cancel   help   << back   next >>   finish

- Here is a sample of what the RMF contains:

```
-- Relational Mapping File Generated By NatMiner on Mon Jun 26 19:39:43 EDT 2006
-- For Converted FDT FILE_094_DB_001
FDT=FILE_094_DB_001
[tables]
1,COPU94CO
[fields]
AA,1,0,N,NUMBER,Y
AB,1,0,A,VARCHAR2,N
AC,1,0,A,VARCHAR2,N
AD,1,0,A,VARCHAR2,N
AE,1,0,D,DATE,N
AF,1,0,D,DATE,N
AG,1,0,A,VARCHAR2,N
AH,1,0,N,NUMBER,Y
AI,1,0,N,NUMBER,Y
AJ,1,0,N,NUMBER,Y
AK,1,0,N,NUMBER,Y
AL,1,0,N,NUMBER,Y
AM,1,0,A,VARCHAR2,N
AN,1,0,N,NUMBER,Y
AO,1,0,N,NUMBER,Y
AP,1,0,N,NUMBER,Y
AQ,1,0,N,NUMBER,Y
AR,1,0,N,NUMBER,Y
AS,1,0,A,VARCHAR2,N
AT,1,0,N,NUMBER,Y
AU,1,0,N,NUMBER,Y
AV,1,0,A,CHAR,N
AW,1,0,D,DATE,N
AX,1,0,D,DATE,N
AY,1,0,N,NUMBER,Y
AZ,1,0,N,NUMBER,Y
A0,1,0,N,NUMBER,Y
A1,1,0,A,VARCHAR2,N
A2,1,0,N,NUMBER,Y
A3,1,0,A,CHAR,N
A4,1,0,N,NUMBER,Y
A6,1,0,D,DATE,N
A7,1,0,N,NUMBER,Y
A8,1,0,D,DATE,N
A9,1,0,N,NUMBER,Y
BA,1,0,D,DATE,N
BB,1,0,A,VARCHAR2,N
BC,1,0,N,NUMBER,Y
BD,1,0,N,NUMBER,Y
BE,1,0,N,NUMBER,Y
BF,1,0,A,VARCHAR2,N
BG,1,0,A,VARCHAR2,N
BH,1,0,A,VARCHAR2,N
BI,1,0,N,NUMBER,Y
BJ,1,0,N,NUMBER,Y
BK,1,0,N,NUMBER,Y
BL,1,0,N,NUMBER,Y
BM,1,0,N,NUMBER,Y
BN,1,0,A,VARCHAR2,N
BO,1,2,N,NUMBER,Y
BP,1,0,N,NUMBER,Y
BQ,1,0,N,NUMBER,Y
BR,1,0,A,VARCHAR2,N
BS,1,0,N,NUMBER,Y
```