ENABLING THE FUTURE

Natural / Adabas

Migration Solutions

JavNat Process Guide

© 2010 FBD ASSOCIATES INC. COPYING OR DISTRIBUTION OF THIS REPORT IS NOT PERMITTED WITHOUT THE PRIOR WRITTEN CONSENT OF FBD ASSOCIATES INC.

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 Java based web application using the FBD Associates Inc. (FBDA) 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 relational schema generation
- Conversion of Natural source code to Java
- Adabas data conversion and migration
- Testing and user acceptance

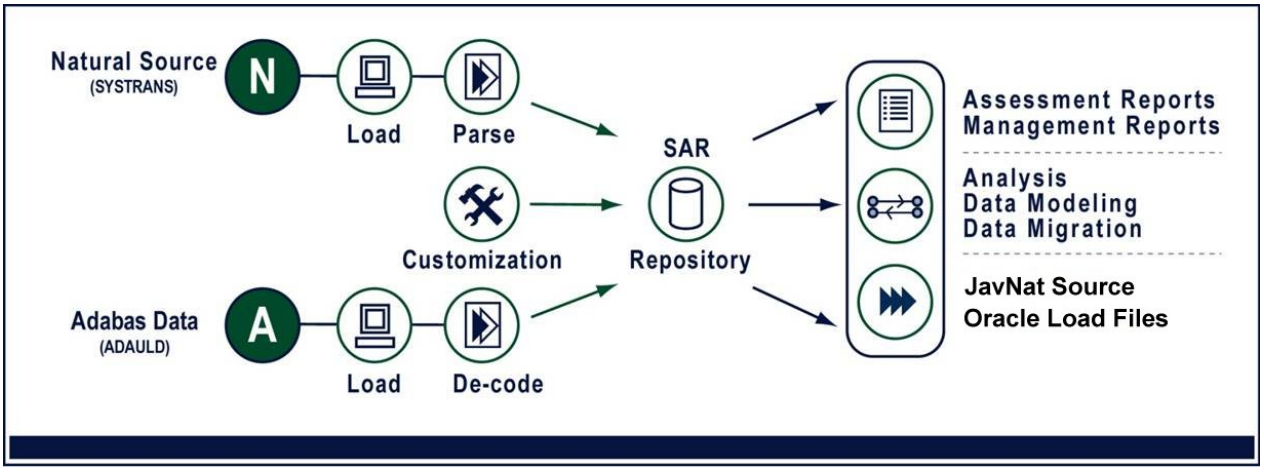


Figure 1 JavNat Process

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.

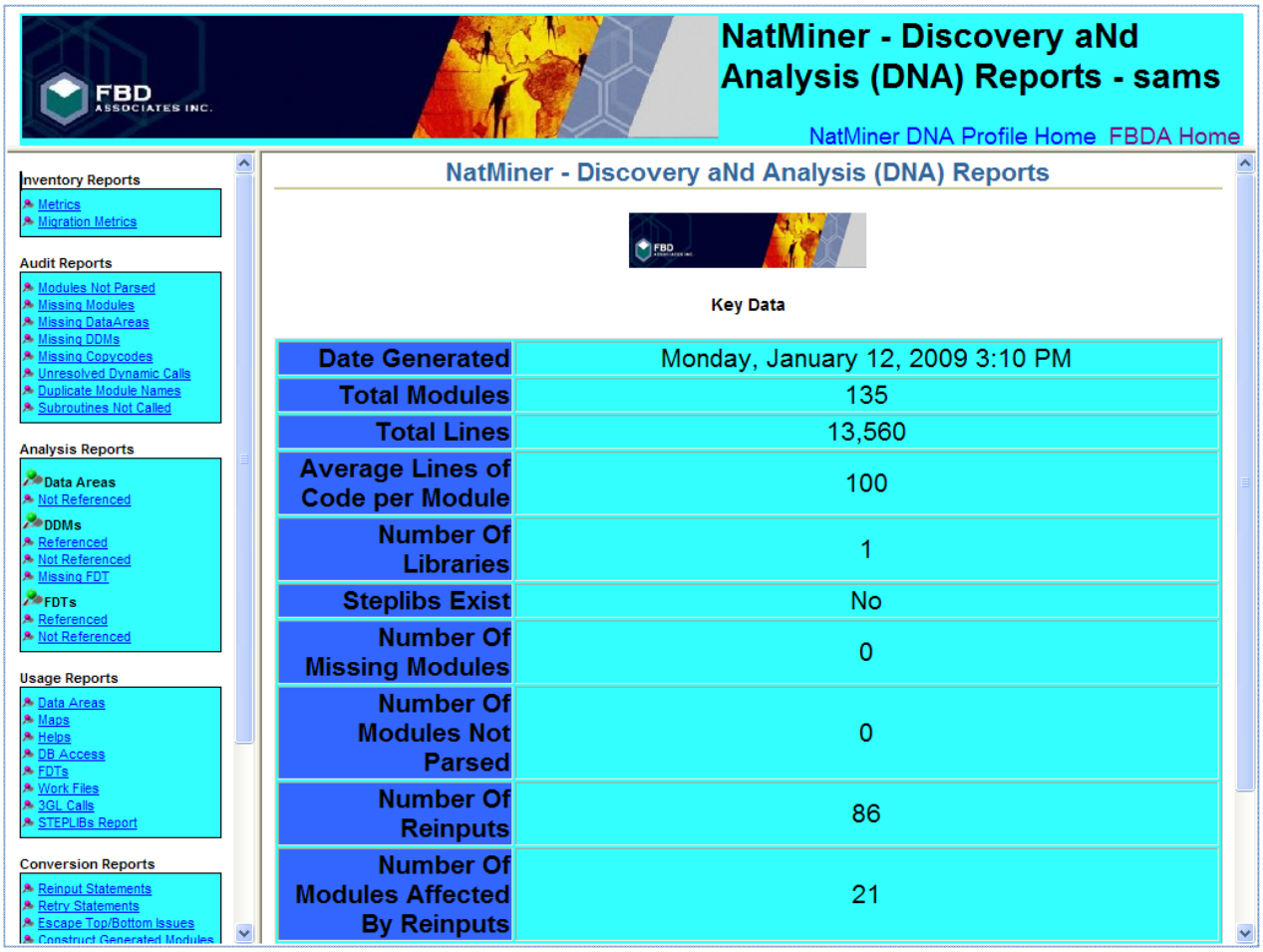
To assist in this activity, FBDA will provide special instructions for source code collection. For details regarding the collection of source code go to the following link:

http://www.fbda.ca/Docs/FBDA%20-%20Code%20Extraction%20Instructions%20V%202.2%20Mar%2005_03.doc

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

3 Parsing and Analysis

The FDTs, DDMs, Maps and all other Natural source objects transmitted in 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 to determine the feasibility of conversion and identify problematic code (e.g.: dynamic code, external calls). Using FBDA's Statistical Normalization and Partitioning Tool (SNAP®), 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 assists a user in performing a number of functions that support application conversion and data migration.

- 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
- 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

The NatMiner component of SNAP® produces comprehensive inventory reports that ensure the completeness and integrity of the code prior to conversion.

4 Partitioning

Using the SNAP® tool and the DNA reports 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.

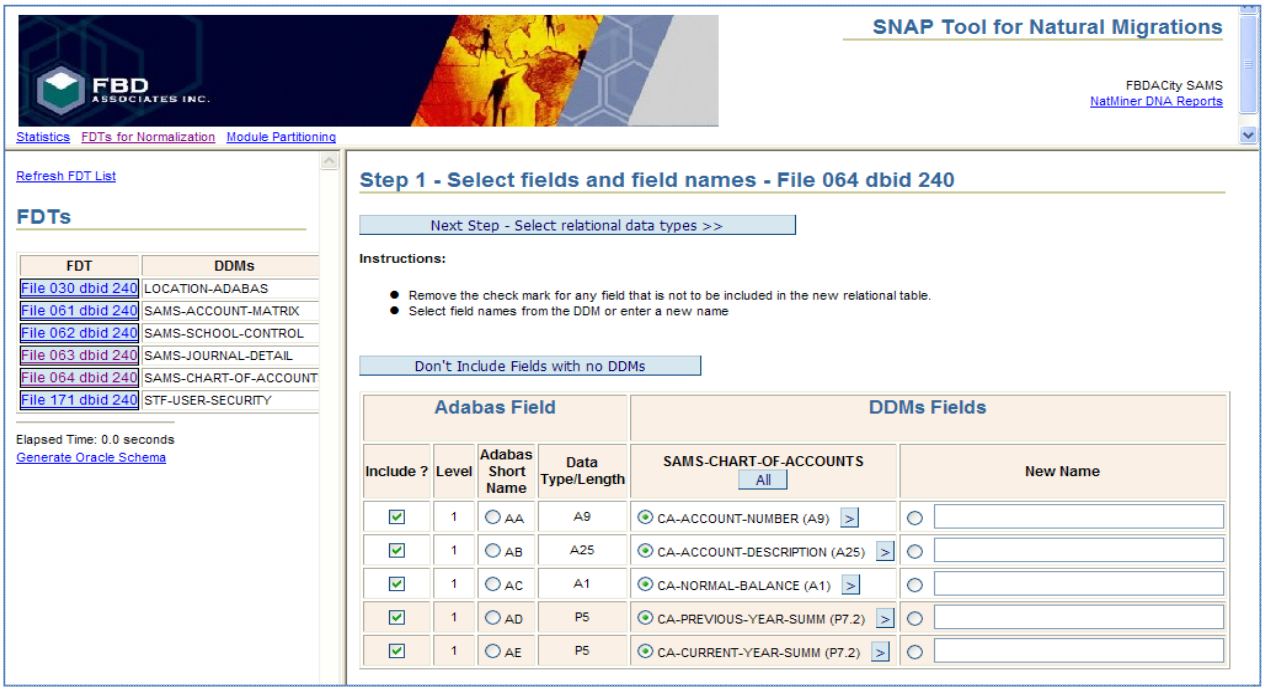
Partitions are designed to follow the normal business procedures. For example, security and support file maintenance would be likely candidates for the first partition. Subsequent partitions would then use and build on the files already created and populated.

In order to achieve the highest quality results in partitioning, it is essential that the customer be involved in this activity 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.

5 Normalization

Once the missing modules are accounted for and the information in SAR is deemed to be complete and timely, the next step is normalization of the FDTs to create a relational schema. The customer has the option of specifying field and table names that will be used in the creation of the new schema, as well as some ability to change data types and sizes. While changes may be provided in any format (Excel, CSV etc.), they are entered into SAR using FBDA's SNAP® tool.



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 1 DDM for the FDT.

Unless the complexity of the FDT-DDM relationship causes a conflict (e.g. 2 DDMs have different names for the same ADABAS field), the JavNat field and table names generated correspond to the DDM names as a default. Where an ADABAS field has no corresponding field in a DDM, its name is built from the Adabas short name. 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.

By default, the relational tables 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, by default 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 might 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. Under

certain circumstances, SQL can be used to copy the functionality of the SP/SB, eliminating the need for a physical field.

- 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
- PL/SQL script to build sub/superdescriptor triggers
- SQL to generate ISN sequence

The generation of the DDL is done online and takes seconds. It can then either be saved for later use, or copied and pasted directly into SQL Plus to create the database tables, indexes and triggers.

6 Source Code Conversion

6.1 Automated conversion to JavNat

Once normalization is completed the source code can be converted. In order to ensure the generation of the correct DML it is essential that the FDT normalization details be stored in SAR before proceeding. This is done by clicking on the button Save Relational Data in the last step of normalization when one is satisfied with the results.

The JavNat source code conversion strategy is based on three key components as follows:

NatLogic Base Classes

The NatLogic base classes are a collection of custom Java methods that implement the Java equivalent functionality of the Natural language syntax. In general each Natural statement is mapped to a Java method on a one to one basis. The JavNat methods are designed to give the converted code a Natural "look and feel."

Support Base Classes

The Support base classes are a collection of custom Java classes to provide the Java equivalent functionality for certain Natural runtime components and features. Some examples are:

- NatSession - maintains and controls the JavNat runtime environment.

- NatSysVar - manages the system variables such as *DATX and *COUNT.
- NatStackItem - manages items passed on the stack
- NatWorkFile - manages the disk I/O for the Natural Read/Write Workfile statements
- NatDDM - maps View structures to the relational schema.

Translation Engine

A SAR driven tool (Natural2Java Converter) translates each Natural module source code, on a statement-by-statement basis, to a functionally equivalent Java class using methods of the NatLogic base class and native Java statements to create Java logic that will execute exactly as it was executed in the Natural environment. Each Natural source module is converted to a Java class that extends the NatLogic base class.

The new Java classes corresponding to the Natural program modules use the NatDDM base classes to generate SQL statements that are executed through JDBC to the relational schema. Logic is generated to read data from the relational schema and store it back into the Natural view structure in the same way that Natural functions. This applies to programs, subprograms, helproutines, maps, LDAs PDAs and GDAs.

Within the support base classes there are methods defined to handle each Natural system variable. These methods perform the data manipulation necessary to obtain the required results. For example, a method called sDATN will return the system date in the format YYYYMMDD.

Appendix B provides a more detailed description of the translation engine operations.

Appendix C shows an example of a NATURAL program and the equivalent JavNat output by the converter.

6.2 Manual Remediation

The converted JavNat programs, subprograms and subroutines will sometimes require some manual remediation in order to compile and function as designed. The NATURAL 'REINPUT' statement, as well as a handful of other situations will always require some manual resolution. The manipulations required to ensure a successful compilation and functioning of these statements are known and documented. This effort will vary depending on the application, and our experience has been that on average this may take 15 minutes per module.

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

7 Data Migration

JavNat also provides tool-supported data migration from Adabas production files to the relational tables. An automated process that can be managed using NatMiner® 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. JavNat can provide automated support for activities such as:

- Validation that a field specified as a date in Adabas and stored as an A6 actually contains valid YYMMDD occurrences and uses lo-values or blanks as a null value.

- Validation of implied foreign key existence and constraint checks.

- Mapping an Adabas A6 date field to a relational date field with blank occurrences replaced with NULL.

- Mapping an Adabas SSN representation to an appropriate relational representation.

- Expansion or replacement of state, region or status codes.

Performance

All application platforms and databases have specific issues when it comes to performance and customization. Particular performance issues that appear during testing can be tuned using a number of methods:

With the object-oriented nature of Java, cosmetic and functional changes can be made to many of the JavNat classes to customize the application where needed.

Customers are provided a complete set of source code to allow for in-house customization or can contract for this service on an as required basis. Customization methods might include:

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

- Changing the functionality of a particular JavNat I/O method for better performance.

- Adding new JavNat I/O methods to handle specific database access situations.

- Replacing JavNat methods with native Java JDBC calls where needed.

- Changing or adding JavNat methods to improve logic.

- Replacing old Natural-like business logic with new object oriented business rule concepts.

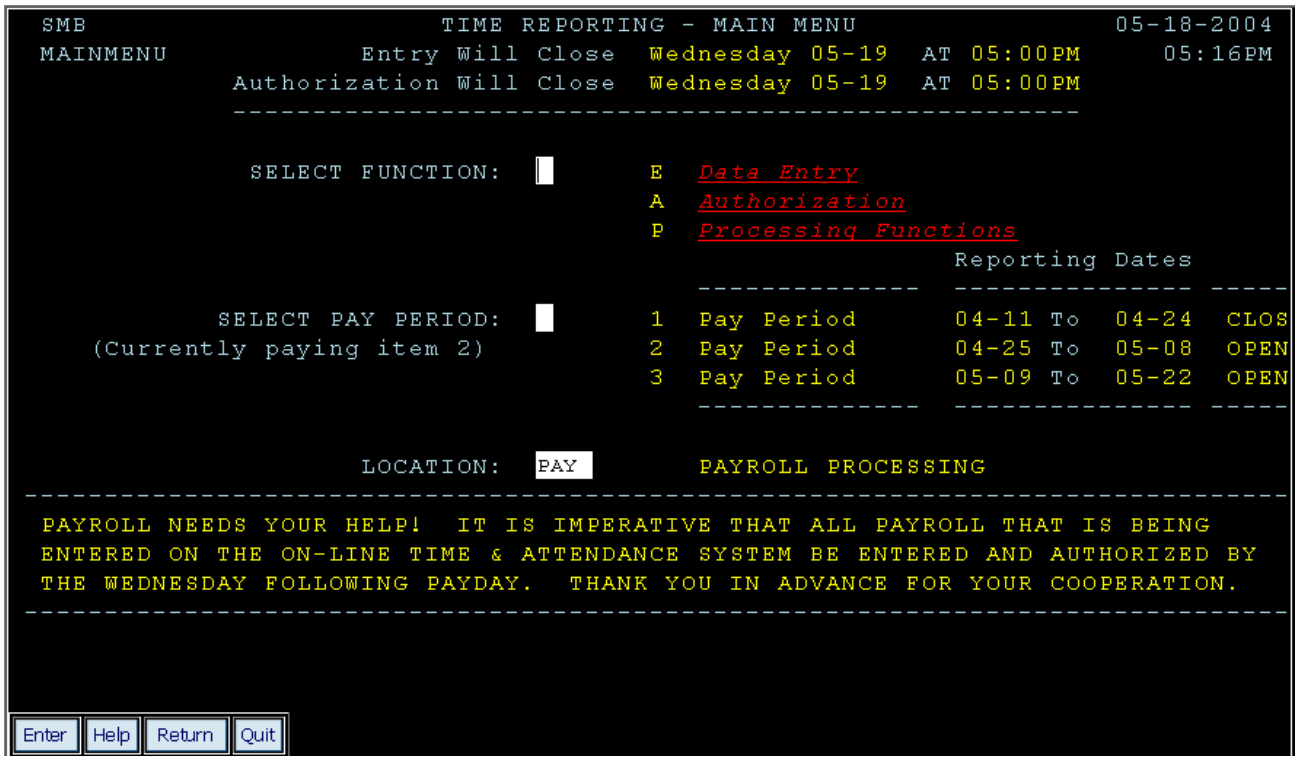
- Re-coding sections in native Java statements.

8 Application Testing

The JavNat conversion process is based on a strict statement to statement conversion approach. As a consequence there is a one-to-one correspondence between converted JavNat 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 JavNat 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 a JavNat 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 JavNat and the NATURAL environment as follows:

8.1.1 Unit test environment

This environment will not be too closely controlled. It is for unit testing and since the testers may not be completely familiar with the application, the data entered may not always make perfect sense. The purpose of these tests is to determine whether each screen seems to function as it should, and it is not intended for system testing.

8.1.2 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. When 'parallel' testing is discussed below it is with this environment in mind.

Since the snapshot of the data will most likely be out of date, the **system date** can be easily set in the JavNat environment to correspond to the date of extraction. Of course this must also be done in the NATURAL environment.

See Appendix "D" for a sample test script.

8.2 Data Conversion

Verification of the data conversion would involve doing an automated comparison between the converted relational tables and the original data.

To accomplish this, the client would provide a dump (flat file) 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 FBDA tools. These two files can then be compared using standard 'diff' software and the results reviewed.

8.3 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. It should exercise every par of the system.

Verification should be done through a regular comparison of:

- database contents
- printed outputs (reports)
- output (work) files produced

8.3.1 Macro level

At the end of a day (or cycle or whatever unit is desirable) of testing, the data comparison should be done between the ADABAS and relational files and any other outputs produced.

8.3.2 Micro level

Note should 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

8.4 Multi-user testing

Once unit testing has been done and the initial system testing has flushed out most of the bugs, the application should be subjected to multiple users working simultaneously. As early as possible tests that involve 2 or more users at a time should be performed. This type of testing should be scheduled as partitions are rolled out, not deferred until the project is completed.

8.5 Problem reporting

It is the responsibility of the tester to report all problems. Obviously *anything* that crashes the system or that is blatantly wrong must be reported as a bug. Apart from that, when odd behavior is noted, it should first be established that this behavior is different from the existing NATURAL application.

- If it is different, then report it as a bug and explain what the correct behavior should be.
- If it is the same but still felt to be a 'bug' then report it as an enhancement and describe what the 'correct' behavior should be.
- Each problem (bug, enhancement, issue) reported should be assigned a priority.
- Each problem reported should be assigned a severity.

It is the responsibility of FBDA to respond with:

- An estimate of the time it will take to address the problem
- A description of the module(s) or other software affected

- A resolution to the problem

As long as there are any unresolved problems, it is recommended that a weekly meeting be held with the specific purpose of exploring these problems, to ensure that the issues are understood by all parties.

9 Appendix "A": External Interfaces

9.1 Batch JCL

Any JCL used to submit batch modules will need to be modified by hand to a new XML based "JCL." This XML will be used by JavNat base classes to pass sequential file and printer information to the converted JavNat classes as JCL did in the Natural application.

Any non-Natural JCL steps within the JCL (sorts, etc.) will need to be re-written using a custom Java class or a 3rd party product.

NATRJE functionality will be replicated by a custom Java class that will be used to submit XML based "JCL" as a batch job.

9.2 Work Files

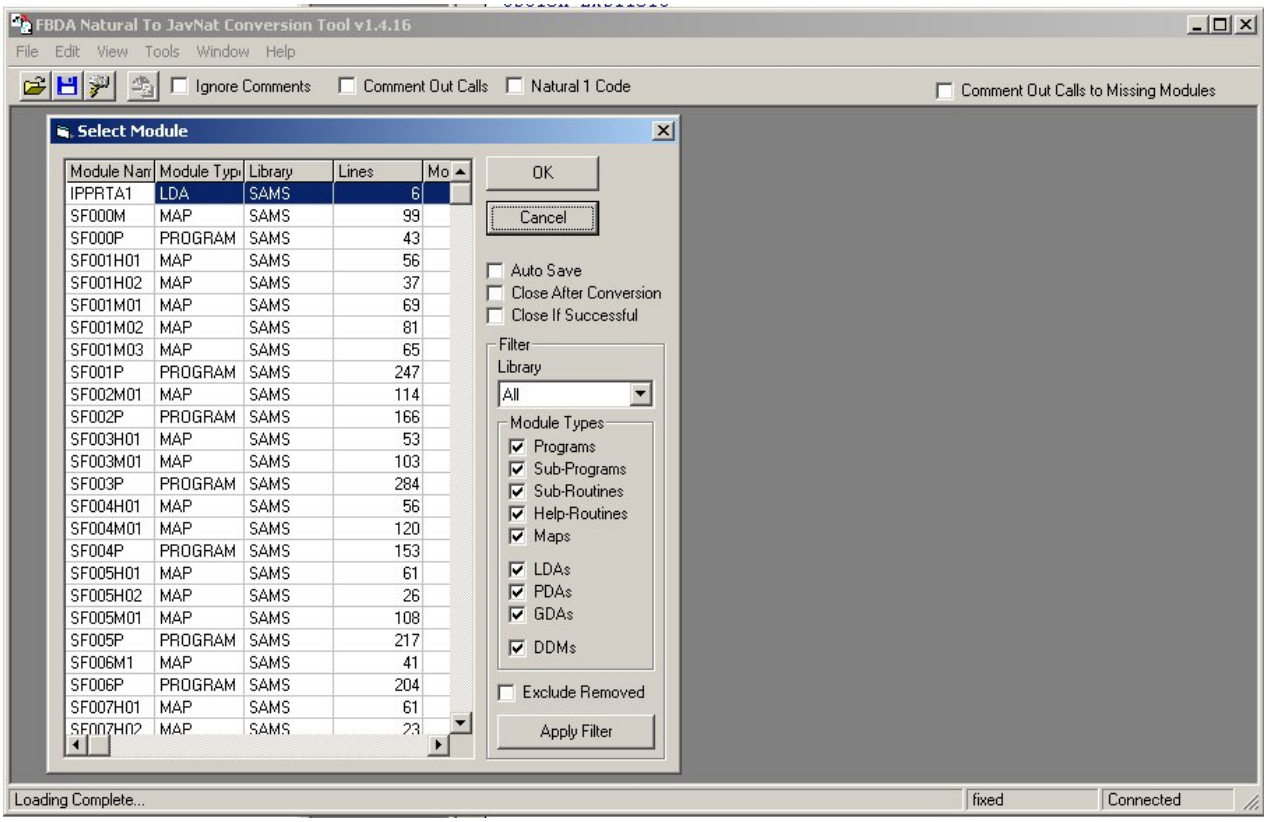
The converted application will have a special Java class which provides the equivalent functionality as the mainframe work files via an access to the target environment file system.

10 Appendix "B": Natural2Java Converter

10.1 Natural2Java Converter

This component is used to convert all the source code (ddms, data areas, maps, executables) into "JavNat®". Section 6.1 above has a summary description of the converter design philosophy. This appendix provides a more detailed description of the converter operations.

The converter is run using a user interface that allows its user a number of options.



- Ignore Comments: The default is to preserve all comments in conversion. When checked, no comments are converted.

- Comment Our Calls: When checked, all Fetch and Callnat statements are commented out in the converted code. For example:

  //FBDA-BLOCKStackTop().Add($PASSAREA_1).Add($PASSAREA_2).Add($PASSAREA_3);
  //FBDA-BLOCKFetch(new pPROG1());

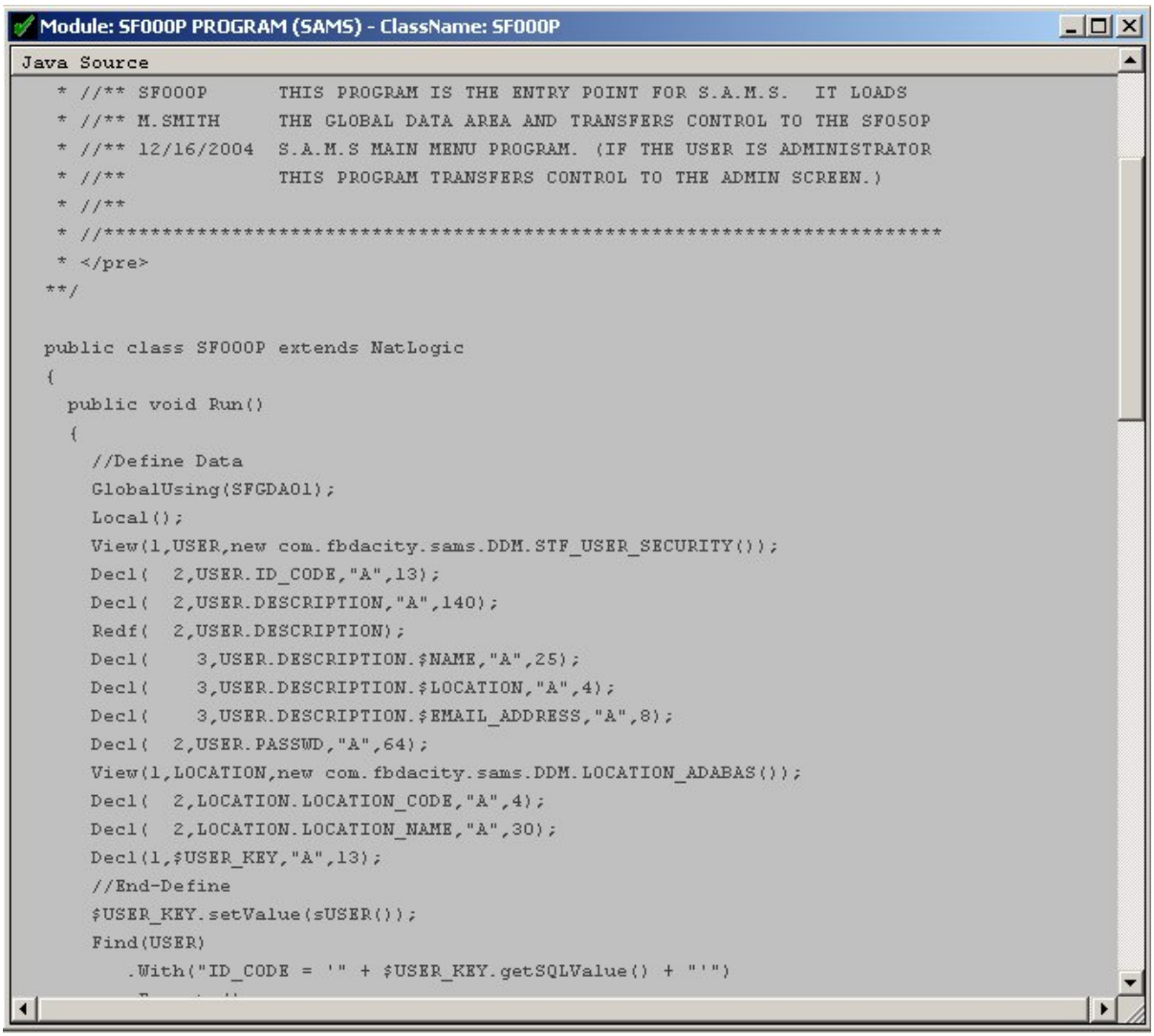
  This is a consideration when compiling and testing. Until the program "PROG1" is successfully compiled, no module that calls it will compile. A strategy is to comment out these calls, and uncomment them as modules are compiled and ready for testing.

- Natural 1 Code: Used to convert Reporting mode code.

In addition to these 'global' settings, there are more options to choose from in the Module Selection window. These are self-explanatory.

Modules can be converted one at a time or as a block. When an issues is encountered (a statement not yet handled, for example), details are reported in a "Conversion Log" window.

Below is an example of a converted module.


Module: SF000P PROGRAM (SAMS) - ClassName: SF000P
Java Source
* //** SF000P THIS PROGRAM IS THE ENTRY POINT FOR S.A.M.S. IT LOADS
* //** M.SMITH THE GLOBAL DATA AREA AND TRANSFERS CONTROL TO THE SF050P
* //** 12/16/2004 S.A.M.S MAIN MENU PROGRAM. (IF THE USER IS ADMINISTRATOR
* //** THIS PROGRAM TRANSFERS CONTROL TO THE ADMIN SCREEN.)
* //**
* //*****
* </pre>
**/
public class SF000P extends NatLogic
{
public void Run()
{
//Define Data
GlobalUsing(SFGDA01);
Local();
View(1,USER,new com.fbdacity.sams.DDM.STF_USER_SECURITY());
Decl(2,USER.ID_CODE,"A",13);
Decl(2,USER.DESCRIPTION,"A",140);
Redf(2,USER.DESCRIPTION);
Decl(3,USER.DESCRIPTION.$NAME,"A",25);
Decl(3,USER.DESCRIPTION.$LOCATION,"A",4);
Decl(3,USER.DESCRIPTION.$EMAIL_ADDRESS,"A",8);
Decl(2,USER.PASSWD,"A",64);
View(1,LOCATION,new com.fbdacity.sams.DDM.LOCATION_ADABAS());
Decl(2,LOCATION.LOCATION_CODE,"A",4);
Decl(2,LOCATION.LOCATION_NAME,"A",30);
Decl(1,$USER_KEY,"A",13);
//End-Define
$USER_KEY.setValue(sUSER());
Find(USER)
.With("ID_CODE = '" + $USER_KEY.getSQLValue() + "'")

11 Appendix "C": Sample of converted JavNat code

NATURAL

```
************************************************************
** PROGRAM NUMBER: SF050P
** PROGRAM NAME  : S.A.M.S. MAIN MENU PROGRAM
** DATE CREATED  : 8/01/98
** PROGRAMMER    : JOHN SMITH
** DESCRIPTION   : THIS IS THE MAIN MENU PROGRAM FOR THE
**                 SCHOOL ACCOUNTING SYSTEM.
**
************************************************************



DEFINE DATA
GLOBAL USING SFGDA01
LOCAL
01 SCHCTRL VIEW OF SAMS-SCHOOL-CONTROL
  02 SC-SCHOOL-NUMBER
  02 SC-CURRENT-MONTH
  02 SC-CURRENT-YEAR
  02 SC-SCHOOL-TYPE
  02 SC-BEGIN-FLAG
  02 SC-RECEIPT-NUMBER
  02 SC-CHECK-NUMBER
1 USER VIEW OF STF-USER-SECURITY
  2 ID-CODE
  2 DESCRIPTION
**
1 LOCATION VIEW OF LOCATION-ADABAS
  2 LOCATION-CODE
  2 LOCATION-NAME
1 #USER-KEY (A13)
1 #DESCRIPTION (A140)
1 REDEFINE #DESCRIPTION
  2 #JUNK(A3)
  2 #FALL-INDICATOR(A1)
1 REDEFINE #DESCRIPTION
  2 #NAME(A25)
  2 #LOCATION(A3)
1 #PROGRAM (A8)
1 #HEADER1 (A60)
1 #HEADER2 (A58)
1 #SCHOOL-CODE (A3)
1 #SCHOOL-NAME (A30)
1 #CNTL      (C)
1 #SELECTION (A2)
1 REDEFINE #SELECTION
  2 #PERIOD (A1)
1 CDGETDCA
  2 #DIRECT-COMMAND (A60)
END-DEFINE



SET CONTROL 'MB'
SET CONTROL 'YN'
SET CONTROL 'YB'
SET KEY ALL
SET KEY
PF1 =  HELP
```

JavNat

```
package com.fbdacity.sams.SAMS;

import fbda.javnat.*;

/**
 * JavNat implementation of PROGRAM SF050P from library SAMS.
 *
 * <p><pre>
 *
//************************************************************
*  //** PROGRAM NUMBER: SF050P
//** PROGRAM NAME  : S.A.M.S. MAIN MENU PROGRAM
//** DATE CREATED  : 8/01/98
//** PROGRAMMER    : JOHN SMITH
//** DESCRIPTION   : THIS IS THE MAIN MENU PROGRAM FOR THE
//**                 SCHOOL ACCOUNTING SYSTEM.
//**
//************************************************************
* </pre>
**/

public class SF050P extends NatLogic
{
  public void Run()
  {
    //Define Data
    GlobalUsing(SFGDA01);
    Local();
    View(1,SCHCTRL,new
com.fbdacity.sams.DDM.SAMS_SCHOOL_CONTROL());
    Decl(  2,SCHCTRL.SC_SCHOOL_NUMBER,"A",3);
    Decl(  2,SCHCTRL.SC_CURRENT_MONTH,"A",2);
    Decl(  2,SCHCTRL.SC_CURRENT_YEAR,"A",4);
    Decl(  2,SCHCTRL.SC_SCHOOL_TYPE,"A",1);
    Decl(  2,SCHCTRL.SC_BEGIN_FLAG,"A",1);
    Decl(  2,SCHCTRL.SC_RECEIPT_NUMBER,"N",6);
    Decl(  2,SCHCTRL.SC_CHECK_NUMBER,"N",6);
    View(1,USER,new
com.fbdacity.sams.DDM.STF_USER_SECURITY());
    Decl(  2,USER.ID_CODE,"A",13);
    Decl(  2,USER.DESCRIPTION,"A",140);
    View(1,LOCATION,new
com.fbdacity.sams.DDM.LOCATION_ADABAS());
    Decl(  2,LOCATION.LOCATION_CODE,"A",4);
    Decl(  2,LOCATION.LOCATION_NAME,"A",30);
    Decl(1,$USER_KEY,"A",13);
    Decl(1,$DESCRIPTION,"A",140);
    Redf(1,$DESCRIPTION);
    Decl(  2,$DESCRIPTION.$JUNK,"A",3);
    Decl(  2,$DESCRIPTION.$FALL_INDICATOR,"A",1);
    Redf(1,$DESCRIPTION);
    Decl(  2,$DESCRIPTION.$NAME,"A",25);
    Decl(  2,$DESCRIPTION.$LOCATION,"A",3);
    Decl(1,$PROGRAM,"A",8);
    // NOT USED Decl(1,$HEADER1,"A",60);
    // NOT USED Decl(1,$HEADER2,"A",58);
    Decl(1,$SCHOOL_CODE,"A",3);
    Decl(1,$SCHOOL_NAME,"A",30);
    Decl(1,$CNTL,"C");
    Decl(1,$SELECTION,"A",2);
    Redf(1,$SELECTION);
    Decl(  2,$SELECTION.$PERIOD,"A",1);
    // NOT USED Decl(1,CDGETDCA);
    // NOT USED Decl(  2,CDGETDCA.$DIRECT_COMMAND,"A",60);
    //End-Define
    SetControl("MB");
    SetControl("YN");
    SetControl("YB");
    SetKeyAll();
    PF1.setKeyHelp();
```

```
  ENTR  NAMED 'ENTER'
  PF3   NAMED 'QUIT'
MOVE *PROGRAM TO #PROGRAM
MOVE *USER TO #USER-KEY
FIND IN FILE USER WITH ID-CODE = #USER-KEY




  MOVE DESCRIPTION TO #DESCRIPTION
END-FIND

FIND IN LOCATION WITH LOCATION-CODE = ##SCHOOL




  MOVE LOCATION-NAME TO #SCHOOL-NAME
  MOVE LOCATION-NAME TO ##SCHOOL-NAME
END-FIND


IF ##USER-LOCATION NE '509' AND *USER NE 'SAMSADM'
  MOVE (AD=P CD=BL) TO #CNTL
END-IF


RESET ##MSG
REPEAT
  MOVE ##SCHOOL TO #SCHOOL-CODE
  MOVE ##SCHOOL-NAME TO #SCHOOL-NAME
  RESET #SELECTION



  INPUT WITH TEXT ##MSG MARK *#SELECTION USING MAP
'SF050M'
  MOVE ' ' TO ##MSG
/*

  IF *PF-KEY = 'PF3' TERMINATE END-IF
/*


  IF #PERIOD = '.'
    FETCH 'SF050P'
  END-IF


/*
  FIND IN LOCATION WITH LOCATION-CODE = #SCHOOL-CODE




    IF NO RECORDS FOUND
      COMPRESS 'LOCATION' #SCHOOL-CODE
        '-IS NOT ON THE LOCATION FILE' INTO ##MSG
      REINPUT FULL WITH TEXT ##MSG
    END-NOREC




    MOVE LOCATION-NAME TO #SCHOOL-NAME
    MOVE LOCATION-NAME TO ##SCHOOL-NAME
    MOVE #SCHOOL-CODE TO ##SCHOOL
  END-FIND
/*



  DECIDE ON FIRST #SELECTION

    VALUE 'ET'
      PERFORM CONTROL
```

```
    ENTR.setNamed("ENTER");
    PF3.setNamed("QUIT");
    $PROGRAM.setValue(sPROGRAM());
    $USER_KEY.setValue(sUSER());
    Find(USER)
       .With("ID_CODE = '" + $USER_KEY.getSQLValue() + "'")
       .Execute();
    while(Next(USER))
    {
      $DESCRIPTION.setValue(USER.DESCRIPTION);
    } //FIND USER

    Find(LOCATION)
       .With("LOCATION_CODE = '" +
SFGDA01.MPS.$$SCHOOL.getSQLValue() + "'")
       .Execute();
    while(Next(LOCATION))
    {
      $SCHOOL_NAME.setValue(LOCATION.LOCATION_NAME);

SFGDA01.MPS.$$SCHOOL_NAME.setValue(LOCATION.LOCATION_NAME);
    } //FIND LOCATION

    if(SFGDA01.MPS.$$USER_LOCATION.ne("509") &&
sUSER().ne("SAMSADM"))
    {
      $CNTL.setValue(CV().AD("P").CD(BL));
    }
    SFGDA01.MSG_INFO.$$MSG.Reset();
    do
    {
      $SCHOOL_CODE.setValue(SFGDA01.MPS.$$SCHOOL);
      $SCHOOL_NAME.setValue(SFGDA01.MPS.$$SCHOOL_NAME);
      $SELECTION.Reset();
      REINPUT_LOOP0: do { //Beginning of Reinput Loop

InputUsingMap(SF050M).WithText(SFGDA01.MSG_INFO.$$MSG).Mark($S
ELECTION).Show();
      SFGDA01.MSG_INFO.$$MSG.setValue(" ");
    //
      if(sPF_KEY().eq("PF3"))
      {
        Terminate(0);
      }
    //
      if($SELECTION.$PERIOD.eq("."))
      {
        Stack().ReleaseStack();
        Fetch(new SF050P());
      }
    //
      Find(LOCATION)
         .With("LOCATION_CODE = '" +
$SCHOOL_CODE.getSQLValue() + "'")
         .Execute();
      while(Next(LOCATION) || NoRecordsFoundPending(LOCATION))
      {
        if(NoRecordsFound(LOCATION))
        {
          Compress().Comp(
             "LOCATION").Comp(
             $SCHOOL_CODE).Comp(
             "-IS NOT ON THE LOCATION FILE")
             .Into(SFGDA01.MSG_INFO.$$MSG);
          Reinput(SFGDA01.MSG_INFO.$$MSG).Full();
          continue REINPUT_LOOP0; //Reinput Support
        }
        $SCHOOL_NAME.setValue(LOCATION.LOCATION_NAME);

SFGDA01.MPS.$$SCHOOL_NAME.setValue(LOCATION.LOCATION_NAME);
        SFGDA01.MPS.$$SCHOOL.setValue($SCHOOL_CODE);
      } //FIND LOCATION

    //
      // Decide On First
      {
        boolean Any79 = false;
        if($SELECTION.eq("ET"))
        {
```

```
      FETCH RETURN 'SF051P'




    VALUE 'QA'
      PERFORM CONTROL
      FETCH RETURN 'SF052P'




    VALUE 'FM'
      FETCH RETURN 'SF053P'




    VALUE 'FD'


      IF ##USER-LOCATION NE '509' AND *USER NE 'SAMSADM'
        REINPUT 'THIS SELECTION IS NOT AVAILABLE TO YOUR
LOCATION'
          MARK *#SELECTION
      ELSE
        FETCH RETURN 'SF060P'
      END-IF




    VALUE 'PR'
      PERFORM CONTROL
      FETCH RETURN 'SF054P'




    VALUE 'CC'
      PERFORM CONTROL
      FETCH RETURN 'SF055P'




    NONE
      REINPUT FULL 'VALID SELECTION CODE MUST BE ENTERED.'
        MARK *#SELECTION
  END-DECIDE


END-REPEAT









************************
DEFINE SUBROUTINE CONTROL
************************
IF #SCHOOL-CODE < '500'
    F1.
  FIND SCHCTRL WITH SC-SCHOOL-NUMBER = #SCHOOL-CODE




    IF NO RECORDS FOUND
```

```
          Any79 = true;
          CONTROL();
          if(ReinputPending()) continue REINPUT_LOOP0; //FBDA
Subroutine CONTROL Contains REINPUT but no INPUT
          FetchReturn(new SF051P());
        }
        else if($SELECTION.eq("QA"))
        {
          Any79 = true;
          CONTROL();
          if(ReinputPending()) continue REINPUT_LOOP0; //FBDA
Subroutine CONTROL Contains REINPUT but no INPUT
          FetchReturn(new SF052P());
        }
        else if($SELECTION.eq("FM"))
        {
          Any79 = true;
          FetchReturn(new SF053P());
        }
        else if($SELECTION.eq("FD"))
        {
          Any79 = true;
          if(SFGDA01.MPS.$$USER_LOCATION.ne("509") &&
sUSER().ne("SAMSADM"))
          {
            Reinput("THIS SELECTION IS NOT AVAILABLE TO YOUR
LOCATION").Mark($SELECTION);
            continue REINPUT_LOOP0; //Reinput Support
          }
          else
          {
            FetchReturn(new SF060P());
          }
        }
        else if($SELECTION.eq("PR"))
        {
          Any79 = true;
          CONTROL();
          if(ReinputPending()) continue REINPUT_LOOP0; //FBDA
Subroutine CONTROL Contains REINPUT but no INPUT
          FetchReturn(new SF054P());
        }
        else if($SELECTION.eq("CC"))
        {
          Any79 = true;
          CONTROL();
          if(ReinputPending()) continue REINPUT_LOOP0; //FBDA
Subroutine CONTROL Contains REINPUT but no INPUT
          FetchReturn(new SF055P());
        }
        if(!Any79)
        {
          Reinput("VALID SELECTION CODE MUST BE
ENTERED.").Mark($SELECTION).Full();
          continue REINPUT_LOOP0; //Reinput Support
        }
      }
      break;  } while(true); // End of REINPUT_LOOP0
    } while(true);
  } /***************************** End Main
******************/

//************************
  /**
   * This method...
  **/
  private void CONTROL()
  {
  //************************
  if($SCHOOL_CODE.lt("500"))
  {
    Find(SCHCTRL)
       .With("SC_SCHOOL_NUMBER = '" +
$SCHOOL_CODE.getSQLValue() + "'")
       .Execute();
    F1:
    while(Next(SCHCTRL) || NoRecordsFoundPending(SCHCTRL))
    {
      if(NoRecordsFound(SCHCTRL))
```

```
      COMPRESS 'NO SCHOOL CONTROL RECORD FOUND FOR
LOCATION -'
**      #SCHOOL-CODE '- PERFORM START-UP' INTO ##MSG
        #SCHOOL-CODE '-' #SCHOOL-NAME INTO ##MSG
      REINPUT FULL WITH TEXT ##MSG
    END-NOREC
  END-FIND
END-IF
END-SUBROUTINE /* CONTROL
END
```

```
      {
        Compress().Comp(
           "NO SCHOOL CONTROL RECORD FOUND FOR LOCATION -
").Comp(
           $SCHOOL_CODE).Comp(
           "-").Comp(
           $SCHOOL_NAME)
           .Into(SFGDA01.MSG_INFO.$$MSG);
        //**      #SCHOOL-CODE '- PERFORM START-UP' INTO ##MSG
        Reinput(SFGDA01.MSG_INFO.$$MSG).Full();
        return;
      }
    } //FIND SCHCTRL
  }
} // End Subroutine CONTROL

  //Data Area and Map Declarations
  SFGDA01 SFGDA01;
  SF050M SF050M = new SF050M();

  public void Initialize()
  {
    SFGDA01 = (SFGDA01) ActiveGlobal(new SFGDA01());

    //AIV Declaration and Registration

    //Undeclared Variables

    //Map References
    setReference($CNTL,SF050M.$CNTL);
    setReference($DESCRIPTION.$NAME,SF050M.$NAME);
    setReference($PROGRAM,SF050M.$PROGRAM);
    setReference($SCHOOL_CODE,SF050M.$SCHOOL_CODE);
    setReference($SCHOOL_NAME,SF050M.$SCHOOL_NAME);
    setReference($SELECTION,SF050M.$SELECTION);
  }

  //Local NatVar Declarations
  private SCHCTRL SCHCTRL = new SCHCTRL("SCHCTRL");
  private class SCHCTRL extends NatView {
  private SCHCTRL(String variableName) { super(variableName);
}
    private NatVar SC_SCHOOL_NUMBER = new NatVar("SC-SCHOOL-
NUMBER");
    private NatVar SC_CURRENT_MONTH = new NatVar("SC-CURRENT-
MONTH");
    private NatVar SC_CURRENT_YEAR = new NatVar("SC-CURRENT-
YEAR");
    private NatVar SC_SCHOOL_TYPE = new NatVar("SC-SCHOOL-
TYPE");
    private NatVar SC_BEGIN_FLAG = new NatVar("SC-BEGIN-
FLAG");
    private NatVar SC_RECEIPT_NUMBER = new NatVar("SC-RECEIPT-
NUMBER");
    private NatVar SC_CHECK_NUMBER = new NatVar("SC-CHECK-
NUMBER");
  }
  private USER USER = new USER("USER");
  private class USER extends NatView {
  private USER(String variableName) { super(variableName); }
    private NatVar ID_CODE = new NatVar("ID-CODE");
    private NatVar DESCRIPTION = new NatVar("DESCRIPTION");
  }
  private LOCATION LOCATION = new LOCATION("LOCATION");
  private class LOCATION extends NatView {
  private LOCATION(String variableName) { super(variableName);
}
    private NatVar LOCATION_CODE = new NatVar("LOCATION-
CODE");
    private NatVar LOCATION_NAME = new NatVar("LOCATION-
NAME");
  }
  private NatVar $USER_KEY = new NatVar("#USER-KEY");
  private $DESCRIPTION $DESCRIPTION = new
$DESCRIPTION("#DESCRIPTION");
  private class $DESCRIPTION extends NatVar {
  private $DESCRIPTION(String variableName) {
super(variableName); }
    private NatVar $JUNK = new NatVar("#JUNK");
```

	private NatVar $FALL_INDICATOR = new NatVar("#FALL-INDICATOR");
	private NatVar $NAME = new NatVar("#NAME");
	private NatVar $LOCATION = new NatVar("#LOCATION");
	}
	private NatVar $PROGRAM = new NatVar("#PROGRAM");
	private NatVar $SCHOOL_CODE = new NatVar("#SCHOOL-CODE");
	private NatVar $SCHOOL_NAME = new NatVar("#SCHOOL-NAME");
	private NatVar $CNTL = new NatVar("#CNTL");
	private $SELECTION $SELECTION = new $SELECTION("#SELECTION");
	private class $SELECTION extends NatVar {
	private $SELECTION(String variableName) { super(variableName); }
	private NatVar $PERIOD = new NatVar("#PERIOD");
	}
	}

12 Appendix "D": Sample User Acceptance test script

Section 1: General Information		
Primary User's Name: Alfred E. Newman		Test Case ID #:
User's Extension #: 763-5462		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:	