Phase 0 HDB DECODES Integration Technical Design and Installation Instructions

Draft Version 1.0   June 23, 2005

Prepared By: Mark Bogner, CADSWES HDB Project Manager

TABLE OF CONTENTS

1.0 INTRODUCTION

The Device Conversion and Delivery System (DECODES) was selected by the Upper Colorado Area Office as the software system that will be utilized to process their satellite telemetry data. The data retrieved from this system is to be placed into the Area Office’s HDB database. Phase 0 was the development effort that would make this implementation happen. This document’s purpose is to technically define how DECODES will be integrated into HDB.

1.1 BACKGROUND

DECODES was originally created for use by the USGS. The DECODES system was comprised of several Java user applications and a relational database management system (RDBMS) known as PostgreSQL. CADSWES has been requested by the Bureau of Reclamation to provide the technical approach necessary to integrate DECODES to the ORACLE HDB database.

1.2 AUDIENCE

The intended audience of this document is either an experienced ORACLE DBA or experienced ORACLE application developers who are thoroughly familiar with the operations of the ORACLE HDB database, unix and unix system commands, and application developers familiar with java and how java code utilizes JDBC and interfaces with ORACLE databases.

2.0 DECODES Java Applications

The intent of this imntegration and implementation effort is that the majority of the DECODES java applications will remain unchanged and as little as possible of the existing java code will be required to be modified. Since Java applications are generally platform independent, the majority of the java applications user interface source code will not require any modifications. There are some modifications to this java software that are required due to the differences in the database implementations of ORACLE and PostgreSQL. The following sub-paragraphs outline the differences discovered so far.

2.0.1 Java ORACLE SEQUENCES UTILIZATION

The DECODES java applications utilize database sequences extensively to create unique record keys in the DECODES database. ORACLE also has sequences capabilities but the SQL code necessary to activate and generate a sequence and to acquire a new sequence is slightly different when compared to PostgreSQL. The DECODES applications planned for this condition and allow for different java classes to be instantiated for sequence key generation. The java software modifications required for to enable the ORACLE sequence capability is to create and compile a java class that acquires sequences utilizing a SQL method that is acceptable for ORACLE. The user of the DECODES application must then alter the DECODES property file and enter the new java class for the SqlKeyGenerator property to utilize the new java class. See the later section entitled “DECODES PROPERTY FILE”, Section 3.0 for additional information regarding the setup and values required for the DECODES property file.

2.0.2 Java ORACLE DATES

Almost all relational databases handle dates and date/time formats differently, so it was no surprise to find that when the DECODES application attempted to place dates in the database, the java applications did not function correctly when attempted against an ORACLE database. The java source code was required to be modified to account for this discrepancy. Additionally, this fix required an additional property to be set with the acceptable date format of the utilized database in the DECODES property file. The user of the DECODES application must then alter the DECODES property file and enter the correct ORACLE date format for their database for the “sqlDateFormat” and the “sqlTimeZone” properties to have the java classes operate correctly. See the later section 3.0 entitled “DECODES PROPERTY FILE” for more information regarding the setup and values required for the property file.

2.1 DECODES DATABASE OBJECTS, PERMISSIONS, ETC…

The majority of the database objects that DECODES applications utilize are standalone tables that have no direct interaction with current HDB tables or HDB capabilities. For these tables, little was modified to get these tables into an ORACLE database. However, some slight DDL syntax modifications were required to some of these database objects were necessary to get ORACLE to accept the DDL.

2.1.1 DECODES SCHEMA CREATION

A separate ORACLE schema within the HDB database was created for the database objects that were only applicable to the DECODES application. This schema will own the DECODES tables and other necessary database objects. The schema name selected for this schema was “DECODES”. The DECODES creation scripts were not modified to create any referential integrity between the tables. They were however modified slightly; specifically an “exit” was added to the end of each creation script so that these scripts could be called from a single driver script called “create_decodes.sh” and operate effectively during the creation of the DECODES database objects. The script that creates the DECODES schema also gave all the proper rights and permissions to this schema so that the DECODES schema could own the necessary database objects and give the correct access permissions to them.

2.1.2 DECODES DATABASE OBJECTS MODIFICATIONS

The DECODES tables creation script that is usually supplied was modified slightly to minimally adhere to ORACLE and HDB standards. The tables and indexes were modified so include the tablespace that the tables and indexes respectively were to reside. PostgreSQL uses a float8 data type for their float columns so in the DDL those columns were modified to float so that the tables creations scripts would work correctly. Since ORACLE generally caches sequences and it was somewhat desireable to maintain sequence integrity, the sequence creation script DDL statements were modified to add the “nocache” option to every statement.

The DDL scripts used to create the DECODES database tables and views can be found in file “createORACLEDecodes.sql”. The script that creates the DECODES sequences can be found in the file named “createORACLEDecodesSequences.sql”.

2.1.3 DATABASE ROLES AND PERMISSIONS

The HDB standard to allow select permissions of all tables and views to public was continued for all the DECODES schema objects. Public synonyms were created for all of the DECODES database objects. The role “DECODES_ROLE” was created to give the insert, update and delete permissions to all of the DECODES tables to this role. Additionally, the DECODES role was granted to the REF_META_ROLE since that role deals with permissions to the metadata tables. Finally the HDB APP_ROLE was granted to the DECODES_ROLE to enable any user that utilizes DECODES the ability to write to HDB tables. This was done because setting of roles at application run time is not a known concept to the DECODES application. Therefore, it was necessary to allow this capability and this capability must be defined and managed utilizing the end user’s default roles. The file named “set_decodes_privs.sql” contains all the necessary DDL to create the public synonyms for the DECODES schema tables, grants select privileges to public on all the DECODES schema tables, and finally, grants select, insert, update, and delete to the role DECODES_ROLE on all the DECODES schema tables.

2.2 HDB SCHEMA MODIFICATIONS

Several HDB tables were modified to integrate the DECODES database and applications into the HDB paradigm. The general approach to make this integration effort succeed was for every HDB table or tables that looked quite similar and was used in HDB for similar functionality to the DECODES tables, create a view of the HDB tables with the same tables names and column names as the DECODES tables but utilizing the HDB tables in the views. Finally, have “INSTEAD OF” triggers on all these views so when DECODES code issued DML statements to these views, the DML was intercepted in the triggers and then the data was handled appropriately and DML statements were issued to the correct HDB tables. The following sections explain in detail the HDB table modifications and the views and triggers that correlate with them.

2.2.1 HDB UNIT TABLE

DECODES utilizes a table called engineering unit. This table has similarities to the HDB table HDB_UNIT. The significant difference between these tables is the primary key. It was decided that the HDB_UNIT table will be modified so that the UNIT_COMMON_NAME would act as the primary key when utilized by DECODES. To accomplish this, the UNIT_COMMON_NAME was sized down to the DECODES UNIT_ABBR size. The column was truncated and reduced to twenty four characters. An additional column “FAMILY” was added to the HDB_UNIT table to correspond with a column of the same name in the DECODES table. The hdb_dimension table is used to identify the HDB version of the measures column within the DECODES engineeringunit table. A view called “UNIT_TO_DECODES_UNIT_VIEW” was created to look identical to the DECODES engineeringunit table. This view uses the newly modified HDB_UNIT table and the HDB_DIMENSION table to replicate exactly the look of the decodes ENGINEERINGUNIT table. Select permissions are granted to public to select from this view and insert, update and delete permissions are granted on this view to the DECODES_ROLE. Control of the DECODES DML directed at this view is by INSTEAD of triggers on this view. The trigger code along with further explanations to the purpose and logic behind the triggers and use of this view are located in file “decodes_engineeringunit.trg”. The delete trigger to this view has been programmed to allow no deletes since a delete to the underlying HDB_UNIT table could have negative consequences to the integrity of the HDB database. On updates, the unit_abbr, measures and family columns in the DECODES applications are permitted to be updated. Inserts to this view are also allowed however, assumptions to other values are needed to allow an insert into the HDB HDB_UNIT table so the trigger defaults certain values not applicable or available from the DECODES application.

2.2.2 HDB SITE TABLE

DECODES deals with deciphering satellite messages from sensors at particular sights. Naturally then, DECODES has a table that stores information regarding a site. The DECODES table for the site information is simply called “SITE”. Additionally, decodes maintains a one-to-many relationship of many site names to a single site based on the type of site the name is being used for (USGS, NWS, etc..). The HDB HDB_SITE table was used for the site data elements that both systems currently possessed. The lat and longi columns in the HDB_SITE table required modification in length so that the two systems were in agreement. Additionally, the description column was increased to a length of 560. This was done with the understanding that the DECODES description, which was originally 800 characters in length, would show a concatenation of the HDB_SITE site_name, a line feed, and then the description. So this new concatenation would add up to 800 characters also. The difficultly surrounding this requirement is that it forces the user of the DECODES application to know this site description rule and he/she must enter the description accurately or an operation error will result when data is attempted to be saved to the database. DECODES has additional data elements pertaining to its sites that HDB does not. To satisfy this requirement, another table DECODES_SITE_EXT was created as a logical extension to the HDB_SITE table and the site_id primary key was used for the primary and foreign key to this table. The creations scripts for this implementation included a script that added rows to this table for any historical sites previously in the HDB database prior to the DECODES implementation. Also to insure data and row integrity for every site, the database trigger for an insert into the hdb_site table also included a creation of a new row into the DECODES_SITE_EXT table. Details of this insert trigger and further explanation of the purposes of the insert and update triggers can be found in the file "decodes_hdb_site.trg". A cascading delete constraint was included in the foreign key to insure that the extension table's row was adequately deleted when a HDB_SITE row was deleted without having to code any additional triggers. A view called "SITE_TO_DECODES_SITE_VIEW" was created to look identical to the DECODES site table. This view uses the newly modified HDB_SITE table, the DECODES_SITE_EXT table, and the HDB_EXT_SITE_CODE table to replicate exactly the look of the DECODES SITE table. Select permissions are granted to public to select from this view and insert, update and delete permissions are granted on this view to the DECODES_ROLE. A public synonym called SITE was created in the HDB schema so that the DECODES application would see the SITE table and would operate correctly. Control of the DECODES DML directed at the SITE table is by INSTEAD of triggers on this view. The trigger code along with further explanations to the purpose and logic behind the triggers and use of this view are located in file "decodes_site.trg". The delete trigger to this view has been programmed to allow deletes only to the HDB_SITE table. On updates, the appropriate columns in the HDB_SITE and the DECODES_SITE_EXT table for their respective data columns in the DECODES applications are permitted to be updated. Inserts to this view are also allowed. However, assumptions for required default values needed for an insert into the HDB HDB_UNIT table weree required. So the insert trigger defaults certain values not applicable or available from the DECODES application.

An additional DECODES capability that has to be duplicated on the HDB database is the ability to identify multiple site names for a single physical site by a site type category. The work performed for the generic mapping capability was ideal for this requirement. Assuming that the generic mapping capabilities tables are populated with all the site names for a particular site, it was relatively simple to duplicate the multiple site names capability in HDB. A view called

“SITE_TO_DECODES_NAME_VIEW” was created to look identical to the DECODES site_name table. This view uses the newly modified HDB_UNIT table and the HDB_DIMENSION table to replicate exactly the look of the decodes NAME table. Select permissions are granted to public to select from this view and insert, update and delete permissions are granted on this view to the DECODES_ROLE. A public synonym called sitename was created for this view so that the DECODES application would operate correctly for queries or dml directed against the sitename table. Control of the DECODES DML directed at the sitename table is by INSTEAD of triggers on this view. The trigger code along with further explanations to the purpose and logic behind the triggers and use of this view are located in file “decodes_sitename.trg”. The delete trigger to this view has been programmed to allow deletes on the record in the HDB HDB_EXT_SITE_CODE table. No updates were programmed and there is no update trigger on this view since no DML that performs updates could be found. Inserts to this view are also allowed and values available from the DECODES application are inserted into the HDB HDB_EXT_SITE_CODE table.

2.2.3 HDB DATATYPE VIEW

DECODES has a DATATYPE table that simply identifies the code for the sensor. This code has an id so the challenge was to match up this id and its code to the HDB HDB_DATATYPE table that is presently utilized to identify the type of data that a sensor will report. The use of the generic mapping tables provided this capability. A view called DATATYPE_TO_DECODES_DT_VIEW was created. This view uses the generic mapping tables HDB_EXT_DATACODE_SYS and HDB_EXT_DATA_CODE to replicate exactly the look of the DECODES DATATYPE table. Select permissions are granted to public to select from this view and insert, update and delete permissions are granted on this view to the DECODES_ROLE. A public synonym called DATATYPE was created for this view so that the DECODES application would operate correctly for query or DML directed against the DATATYPE table. No DML should be issued against this view from the DECODES application so the view instead of triggers were not required.

2.2.4 HDB SITE SEQUENCE

DECODE utilizes sequences exclusively for the generation of its tables primary keys. This presents a problem when generating new rows of HDB site data. Since sites can be entered through DECODES, normal SQL, and the Meta Data application, there has to be a consistent method to generate the site_id column. Since the intent was to modify DECODES code as little as possible, the best solution was to use a sequence to generate the site_id for all applications. To accomplish this a sequence called HDB_SITE_SEQUENCE was created, select permissions on this sequence was granted to the DECODES_ROLE and then a public synonym of the name SiteIdSeq was created so that the DECODES application would see and be able to generate a new site_id from the HDB_SITE_SEQUENCE. The procedure code in HDB procedure pop_pk.spb also required modification so that when this procedure was called for table HDB_SITE form the Metadata application, then the site_id returned would have been one retrieved through the use of the newly created sequence.

2.2.5 HDB DECODES View

Once DECODES has the data ready to place into HDB, a mechanism must be available to do so. A view with an insert instead of trigger will provide that mechanism. The view name is DECODES_HDB_INSERT_VIEW and the insert instead of trigger for this view will take the values and call the ORACLE stored procedure WRITE_TO_HDB. Insert permissions to this view are granted to the DECODES_ROLE role. A public synonym of the same name is also created. This allows the DECODES application code to recognize the view and to issue DML statement against it.

2.2.6 HDB WRITE_TO_HDB Stored Procedure

The procedure WRITE_TO_HDB was written to provide multiple uses for writing data to HDB. In general the stored procedure will take the data passed to it and call the main stored procedure MODIFY_R_BASE_RAW so that the data will be inserted into the R_BASE table. Additionally this stored procedure was written to also handle model data that is not to go into R_BASE but instead the data should be placed into the model tables. Various default values are assigned with in this stored procedure and the DBA at the install site must review and edit this stored procedure so that the default values of the stored procedure agree with the values within the database. The code for this procedure that requires a review and possible modification by the DBA is called "write_to_hdb_.prc". No synonyms or permissions are created for this procedure since the execution of this procedure is within the HDB schema owner.

2.3 HDB DATA MODIFICATIONS

DECODES has several data defaults for correct operation of its application. Most of this data is internal to the DECODES tables that have been left as is. However, since we have duplicated the DECODES table structures via views to existing and modified HDB tables, some of the default data must reside within the HDB tables too. The following paragraphs identify this default data.

2.3.1 GENERIC MAPPING SYSTEM SITE DATA

The DECODES application currently supports sites that are of the following three categories: local, usgs, or nwshb5. These defaults must reside and be placed into the HDB_EXST_SITE_CODE_SYS table.

2.3.2 GENERIC MAPPING SYSTEM SYSTEMS DATA

The DECODES application currently supports the following three data coding systems: epa-code, shef_pe, or hydrstra-code. These defaults must be resident in the HDB_EXT_DATA_CODE_SYS table.

3.0 DECODES PROPERTY FILE SETTINGS

Decodes maintains a property file that allows the configuration of the DECODES to be set dynamically and allow the configuration to be set at run-time. The following paragraphs identify and describe the parameter settings that one must consider to have DECODES function correctly against and installed HDB. The name of the file to set these run time DECODES parameters should adequately identified in the DECODES User’s guide, but presently the file name is “decodes.properties” and in located in the root directory of the DECODES installation, usually ../DECODES directory. Any entry in this file proceeded by one or more of the “#” symbol is considered a comment and will be ignored in this file by the DECODES application.

3.0.1 DatabaseType

Decodes default setup is usually for local xml files. Since HDB will be utilized this parameter must be changed to indicate that the operational database is a relational database. The parameter line must be “DatabaseType=SQL”

3.0.2 EditDatabaseType

Decodes default setup is usually for local xml files. Since HDB will be utilized for this DECODES implementation, this parameter must be changed to indicate that the provisional working database is a relational database. The parameter line must be “EditDatabaseType=SQL”

3.0.3 EDITDatabaseLocation

The Decodes connection software must know where the database is located and its name. The standard ORACLE jdbc url to make this work for an HDB must be a parameter line like: “EditDatabaseLocation=jdbc:oracle:thin:@machine_name:1521:ORACLE_SID”. The machine_name and ORACLE_SID are edited to account for your specific HDB installation.

3.0.4 jdbcDriverClass

The Decodes java code must know what driver classes will be used to connect to the HDB database. For use of the ORACLE drivers the parameter line must be “jdbcDriverClass=oracle.jdbc.driver.OracleDriver”.

3.0.5 SqlKeyGenerator

The Decodes code utilizes sequences to generate unique primary keys for its tables. If another java class has been developed and compiled that accomplishes this for the ORACLE databases, then this parameter must indicate this class. If the java class was named newclass.java, the parameter line would be : “SqlKeyGenerator=decodes.sql.newclass”. This parameter line should reflect the real name of the class and its location in a java jar file.

3.0.6 sqlDateFormat

The Decodes code must know the format that the date will be in when issuing DML commands to the database. The parameter line must be equal to the format of your date/time string ie “sqlDateFormat=DD-MON-YYYY 24HR:MM:SS”.

3.0.7 sqlTimeZone

The Decodes code must know what time zone the HDB database is in. The parameter must be of the form “sqlTimeZone=MST” and the timezone must conform to the database’s time zone.

4.0 INSTALLATION INSTRUCTIONS

The basic installation of the database modifications for the phase0 implementation for DECODES integration is a quite simple process of un-tarring a unix tar file and then executing a supplied installation script. The installation of this application assumes that an HDB database has already been installed and the additional capability for the generic mapping tables has also correctly been issued to the database. Additionally, it is assumed that the DECODES application has been correctly installed and that installation instructions in the DECODES user manual have been accurately followed.

4.1 INSTALLATION FILES

The following paragraph highlight the files that the DBA must make note of when attemping to install the phase 0 iDECODES integration effort. Some of these file may need to be modified for certain installations and other SQL commands are required for each installation.

4.1.0 phase0.tar

The supplied tar file Phase0.tar contains the installation script and all the DDL and DML scripts necessary to install the necessary new and modified database objects. First change your default directory by issued a unix cd command. Un-tar the file using the command: tar –xvf phase0.tar. All the files extracted from the tar file are needed for correctly applying the changes to the HDB database. The following files need special considerations in the installation step.

4.1.1 ojdbc14.zip

The ojdbc14.zip file is an ORACLE supplied library for jdbc connectivity to ORACLE databases. This library is necessary for correct operation and connectivity of the DECODES java code to the HDB database. The installation of DECODEs must insure that this file is contained in the environment CLASSPATH before executing the DECODES run script or the reference to this file must be added within the DECODES batch file. It is recommended that this file be moved from the current directory to a location on the network that makes sense for the particular system. This zip file has been tested to work correctly against ORACLE9i and 10G databases. It has also been reported that this zip file is backward compatible for version seven databases, but this was only tested for certain DMI’s that were being revised.

4.1.2 write_to_hdb.prc

The file “write_to_hdb.prc” is an ORACLE procedure whose purpose is to place data received via the DECODES application into the HDB R_BASE table. This procedure was also written for future development on the calculation processor. Subsequently, this code had to flexible enough to satisfy multiple requirements while still being able to call either the modify_r_base_raw or the modify_m_table_raw stored procedures. The stored procedure was written in such a way as the DBA of the database has the option to change certain default values and criteria for this stored procedure. It is advised that the DBA review these default values and modify where necessary to insure proper operability of this procedure within the HDB database.

4.1.3 create_decodes.sh

The file create_decodes.sh is the script that will perform the necessary changes to an existing HDB to make that database compatible with the DECODES application. This script asks the user for the HDB DBA name and password for the database and then takes over from there and issues all the DDL and DML commands to modify the HDB database. The DBA of the HDB database must issue is the command to exercise this shell script. Normally a ./create_decodes.sh command at the unix prompt will adequately run the script. This script not only issues the database commands but it generates several log files with a “.out” extension that echos the commands issues and the feedback received from the database regarding these commands. Although the script performs a grep for “ERROR” in the .out files, it is still recommended the the DBA review these “.out” files and insure there were no installation errors. Once the errors, if any, have been identified and fixed, the database is ready for use by decodes.

4.1.4 decodes_tbl_changes.sql

The file “decodes_tbl_changes.sql” is the file that contains the majority of the DDL and DML statements necessary for the DECODES implementation. Within this file are some insert statements to the Generic Mapping Tables. The insert statements currently default the agen_id for these tables to a default value of seven (7). This assumes two things, that a seven is a valid agen_id and that these are the defaults that you want. It is advised that these agen_id values be modified to suit your database values.

4.1.5 GENERIC MAPPING Tables

The DECODES phase 0 implementation strategy was to utilize to the most extent possible the generic mapping tables. Specifically, sites must have records in the HDB_EXT_SITE_CODE table in order to see any sites within the DECODES application. The SQL to insert these records will be left to the DBA at installation time, since the needs for site record are installation specific. The generic mapping tables also allow the capability to further define data at the sensor level. Again, it is up to the DBA to decide the extent that an installation may use this capability and modify the table entries appropriately.

4.1.6 Create DECODES users

The ORACLE DBA must now either create specific ORACLE user accounts or modify the existing ORACLE accounts for the individuals that will utilize the DECODES applications. Besides the normal access and rights that an ORACLE account usually gets, the DBA must also assure that these accounts are given the DECODES_ROLE as the account’s default role. This historically was not something that a HDB DBA normally was required to do but the need for individual DECODES accounts, for DECODES to access the HDB database and for the fact that DECODES applications did not dynamically assign any database roles made for this new database requirement.