COSPAS-SARSAT
INTERNATIONAL 406 MHZ
BEACON REGISTRATION DATABASE
(IBRD)

SOFTWARE MAINTENANCE MANUAL

C/S D.002
Issue 1
November 2005

COSPAS SARSAT

INTERNATIONAL 406 MHz BEACON REGISTRATION DATABASE (IBRD)
SOFTWARE MAINTENANCE MANUAL

History

Issue	Revision	Date	Revised Pages	Comments
1		November 2005	All new	Approved CSC-35

LIST OF PAGES

Page #	Date of latest revision	Page #	Date of latest revision	Page #	Date of latest revision
cover	Nov 05	5-1	Nov 05	8-20	Nov 05
i	Nov 05	5-2	Nov 05	8-21	Nov 05
ii	Nov 05	5-3	Nov 05	8-22	Nov 05
iii	Nov 05	5-4	Nov 05	8-23	Nov 05
iv	Nov 05			8-24	Nov 05
v	Nov 05	6-1	Nov 05	8-25	Nov 05
vi	Nov 05	6-2	Nov 05	8-26	Nov 05
		6-3	Nov 05		
1-1	Nov 05	6-4	Nov 05	9-1	Nov 05
1-2	Nov 05	6-5	Nov 05	9-2	Nov 05
		6-6	Nov 05	9-3	Nov 05
2-1	Nov 05			9-4	Nov 05
2-2	Nov 05	7-1	Nov 05		
		7-2	Nov 05	10-1	Nov 05
3-1	Nov 05	7-3	Nov 05	10-2	Nov 05
3-2	Nov 05	7-4	Nov 05	10-3	Nov 05
3-3	Nov 05			10-4	
3-4	Nov 05	8-1	Nov 05	10-5	Nov 05
		8-2	Nov 05	10-6	Nov 05
4-1	Nov 05	8-3	Nov 05	10-7	Nov 05
4-2	Nov 05	8-4	Nov 05	10-8	Nov 05
4-3	Nov 05	8-5	Nov 05		
4-4	Nov 05	8-6	Nov 05	A-1	Nov 05
4-5	Nov 05	8-7	Nov 05	A-2	Nov 05
4-6	Nov 05	8-8	Nov 05		
4-7	Nov 05	8-9	Nov 05		
4-8	Nov 05	8-10	Nov 05		
4-9	Nov 05	8-11	Nov 05		
4-10	Nov 05	8-12	Nov 05		
4-11	Nov 05	8-13	Nov 05		
4-12	Nov 05	8-14	Nov 05		
4-13	Nov 05	8-15	Nov 05		
4-14	Nov 05	8-16	Nov 05		
4-15	Nov 05	8-17	Nov 05		
4-16	Nov 05	8-18	Nov 05		
		8-19	Nov 05		

TABLE OF CONTENTS

Page

LIST OF ANNEXES

LIST OF FIGURES

LIST OF TABLES

page left blank

1. INTRODUCTION

1.1 Purpose

This document provides information to support software maintenance efforts for an International 406 MHz Beacon Registration Database (IBRD). Topics covered include the overall system structure, specific software components, the underlying data structures, detailed module mappings and information on performing software modifications.

1.2 Background

Cospas-Sarsat Participants operate a satellite system capable of detecting and locating distress alert transmissions from radio beacons operating at 121.5, 243 and 406 MHz. The beacon signals transmitted over 121.5 MHz and 243 MHz do not include any identification that can be processed by the receiving stations of the Cospas-Sarsat system. Therefore, there is no operational advantage to registering these types of beacons.

The Cospas-Sarsat 406 MHz system provides search and rescue (SAR) services with distress alerts that include the unique 15-character hexadecimal identification of the transmitting beacon. This beacon identification can be decoded to obtain information including:

a) the type of beacon, i.e. aircraft Emergency Locator Transmitter (ELT), vessel Emergency Position Indicating Radio Beacon (EPIRB) or Personal Locator Beacon (PLB),
b) the country code and identification data which form the unique beacon identification, and
c) the type of auxiliary radio locating (homing) device.

If a beacon is properly registered, the 15-character hexadecimal identification of the beacon can be used to access additional information. Beacon registration databases can provide information of great use to SAR services, including:

a) specific aircraft or vessel identification information,
b) the make/model of aircraft or vessel in distress,
c) communications equipment available, and
d) the number of persons onboard.

Such information can be made available to SAR services only if the required information is provided to the registration authority by the beacon owner/operator.
Registration of 406 MHz beacons is required in accordance with international regulations on SAR established by the International Civil Aviation Organization (ICAO) and the International Maritime Organization (IMO), and registration information must be made available to SAR services on a 24-hour basis. A number of countries have made 406 MHz beacon registration mandatory and maintain national 406 MHz beacon registration databases.

However, despite the clear advantages of registration, a large number of 406 MHz beacons are not properly registered due to a lack of registration facilities in a number of countries. Furthermore, a number of beacon registries do not have 24-hour points-of-contact easily

accessible by SAR services. The IBRD is freely available to users with no access to national registration facilities and to Administrations who wish to avail themselves of the facility to make their national beacon registration data more available to SAR services.

The IBRD provides various levels of access to:

a) beacon owners who wish to register their beacons,
b) Administrations who wish to make registration data available to international SAR services, and
c) SAR services that need to access beacon registration data to efficiently process distress alerts.

Cospas-Sarsat provides the IBRD solely for the purpose of assisting SAR Services in SAR operations and is not intended to fulfil the obligation of National Administrations, as required by IMO and ICAO, to provide a National beacon registration facility.

1.3 Document Organization

Section 2 provides an Overview of the IBRD System.

Section 3 outlines the Software Components that comprise the IBRD System.

Section 4 details Data Structures and other Configuration elements.

Section 5 provides background information on the Java 2 Enterprise Edition (J2EE) paradigm.

Section 6 outlines the deployment and software structure of the Main IBRD Application.

Section 7 details the Mappings between functions and the underlying software modules for the Main IBRD Application.

Section 8 describes various Internal Structures and Supporting Elements used by the various IBRD software components.

Section 9 provides information for performing Software Modifications.

Section 10 details the process of IBRD System Installation.

- END OF SECTION 1 -

2. OVERVIEW

The IBRD System provides a full featured web based capability for storing and querying data pertaining to the registration of 406 MHz distress beacons. This system involves one core application with a number supporting elements and packages. The various software components are discussed at a general level in the next section, and in further detail as appropriate in the sections that follow. The central structural element of the IBRD System is the database, which houses the main registration records table as well as a variety of other tables used for supporting operations such as configuration, logging and the generation of reports.

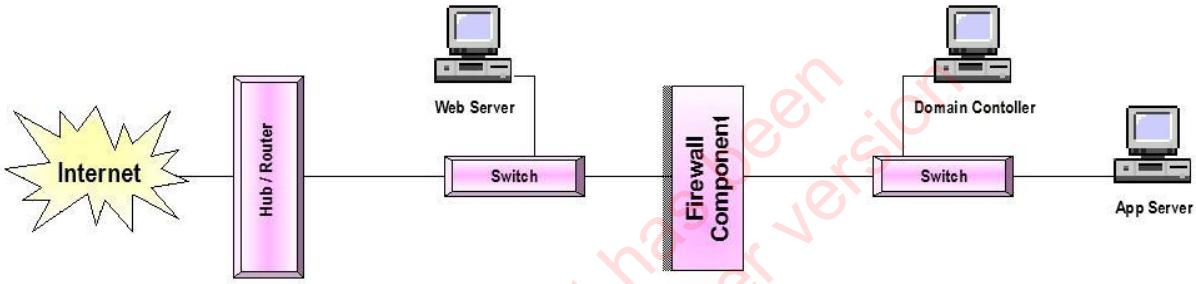

Figure 2.1 General Hardware Structure

In order to provide an appropriate minimum level of security, the hardware that supports the IBRD has the general structure depicted in Figure 1. In fact the actual hardware configuration will likely differ somewhat (e.g., another firewall between the Internet and the Web Server) but the essential elements pertinent to this discussion, the Web Server and App Server computers, are shown.

This layout separates and protects the main components of the IBRD System from the Internet with at least the one Firewall as shown. All critical system functions reside on the Application Server, behind this firewall, while the basic task of handling and fulfilling Internet requests and responses is performed by the Web Server. The only software component residing on the Web Server is the third party package that performs this task, the Apache HTTP Server. Beyond the installation process (discussed in Section 9), very little is said in this document about the Web Server and/or the associated Apache package. The focus for this manual is on all the software elements that reside on Application Server.

In relation to a general overview of the IBRD System, it is perhaps useful to at least briefly discuss the origins of the application software. This software is an adaptation of an existing application, the USA 406 MHz Emergency Beacon Registration Database (RGDB). As such, a significant portion of the underlying structures as well as various naming conventions are inherited from the original USA application software. A useful aside for the sake of explanation is the occasional use of the text “noaa” in internal structures and/or file and module naming conventions. NOAA is the acronym for the “National Oceanographic and Atmospheric Administration”, the division of the USA government where the RGDB is housed.

- END OF SECTION 2 -

page left blank

3. SOFTWARE COMPONENTS

This section introduces the various software components that make up the IBRD System. In general, further details and underlying elements are detailed in the following sections.

3.1 Main IBRD Application

The central component of the IBRD System is a web application that employs Java 2 Enterprise Edition (J2EE) technology and an SQL database to provide a reliable and full featured system for storing and querying data pertaining to the registration of 406 MHz distress beacons.

The Main IBRD Application software involves a fairly complex set of source code that uses most of the tools available in the J2EE paradigm. Classic J2EE features include: native Java code, Enterprise Java Beans (EJB), Standard Java Beans, Java Scripts, Custom Tag Libraries, Java Servlets and Java Server Pages (JSP). The software also uses various other web programming tools including: Hyper Text Markup Language (HTML), eXtensible Markup Language (XML), Cascading Style Sheets (CSS), Properties files and various image files. Although the following sections attempt to describe many of these elements and their usage to some degree, it is assumed that users of this manual either have some background in these areas or will accordingly consult other resources for further information. Sections 5, 6, and 7 focus on the details of these various software elements that make up the Main IBRD Application.

3.2 Supporting IBRD Components

3.2.1 Beacon Decode

The “Beacon Decode” software component provides the specialized function of decoding the Beacon Identification Code used for 406 MHz emergency distress beacons. These beacons employ a 60 bit digital code (translates to 15 hexadecimal characters) which embeds a variety of useful information as briefly discussed in section 1.2 above. For a full explanation of the various encoding protocols and data that may be encoded, the document “<406MHz TECHNICAL BEACON SPECIFICATION…>” (C/S T.001) should be consulted.

For the purposes here, it suffices to understand that this software component takes the 15 character hexadecimal representation of the applicable 60 bits as input, and returns the decoded fields needed by the IBRD application. The software is coded in C++, and is accessed via a dynamic link library (DLL) using the Java Native Interface (JNI). The C++ code itself is actually linked in as a static library in the DLL which is called IBRD_BeaconDecode.dll. More details are provided Section 8.15 below as well in Section 9 with regard to building new software releases.

3.2.2 Purging of Temporary Files

The file archive and purge application (IBRDFileArcPurge.exe) is a simple program that identifies various temporary IBRD system files (e.g. text based log files) and removes old (or aged-out) files based on the date. (Although the ability to archive files is available the configuration for the IBRD generally simply purges files). The folders (or file paths) and the number of days to retain a given type of file is configured using the table named "DbmnFileArcPurgeCfg" which resides in the IBRD database. This application is coded in Microsoft Visual Basic (Version 6.0) and is set up to run as a Windows Scheduled Task. See Section 8.13 for more details.

3.2.3 Database Archival

The table archive and purge application (IBRDArcPurgeTables.exe) takes care of archiving data from various tables in the IBRD database. This program moves "aged out" records to similar tables found in a second database, the IBRDArchive database. The number of days that records are kept in each table in the main IBRD database is configured by setting appropriate values in the SystemCfg table. This application itself is coded in Microsoft Visual Basic (Version 6.0) and is set up to run as a Windows Scheduled Task. The underlying work is performed by a set of SQL Stored Procedures which are called accordingly by the Visual Basic code. See Section 8.14 for more details.

3.2.4 Request for Confirmation Process

The Request for Confirmation Process runs in the background in the form of an operating system batch program named "ProcessTwoYearRequest.bat". The purpose of this process is to generate emails to beacon owners whose records have not been updated in about two or more years. The email message specifically requests these beacon owners to access the IBRD System and confirm or accordingly update the information in the database. The batch file actually runs a Java application using JRun4 tools. The underlying code shares many modules with the Main IBRD Application and resides along with the Main Application code in a web archive file. See Section 8.12 for more details.

3.3 COTS and Open Source Components

The following table is a list of commercial off the shelf (COTS) and open source components used by the IBRD System. Version numbers are those applicable at the time of Acceptance Testing and indicate the minimum recommended for reliable operations.

Name	Version	Type	Description
Microsoft Windows Operating System	2000	COTS	Supplies the computer operating system.
Microsoft SQL Server Standard Edition	2000 (SP3)	COTS	Provides the platform for the IBRD database.
Macromedia JRun	4	COTS	Provides the Java Virtual Machine environment

Name	Version	Type	Description
			that supports the main IBRD application. (includes Macromedia Type IV JDBC Driver for Microsoft SQL Server)
Java SDK	1.4.2	open source	Provides the underlying Java Runtime Environment (JRE) used by JRun4.
Apache HTTP Server	2.0.48+	open source	Provides the HTTP Server – version with SSL support (mod_ssl) is required.
Jacob.jar	1.7	open source	This Java Archive (JAR) package provides a JAva-COm-Bridge (Jacob) which allows calls to COM Automation components from within Java code. It uses the Java Native Interface (JNI) to make calls into the COM and Win32 libraries (used for calling Beacon Decode DLL).
Jacob.dll	1.7	open source	Dynamic Link Library (DLL) associated and used with Jacob.jar to support Win32 library.
Xerces.jar	N/A	open source	This is an Parser which reads in eXtensible Markup Language (XML) and extracts tags and elements accordingly
Log4J-1.2.4.jar	1.2.4	open source	Logging package containing an API that provides detailed context for application error or exception logging.
mail.jar	N/A	open source	JavaMail packages which provides facilities for reading and sending email and supports multiple protocols and service providers including SMTP, IMAP and POP3
activation.jar	N/A	open source	Used in association with the above JavaMail API. The activation.jar contains Java class files as part of the JavaBeans Activation Framework (JAF)
J2EE.jar	N/A	open source	Core Library Package for J2EE applications.

- END OF SECTION 3 -

page left blank

4. DATA STRUCTURES AND CONFIGURATION

The IBRD application relies on an SQL database for of its central purpose which is the storage of beacon registration records. In addition, database tables are used to facilitate various functions such as system configuration and logging. The following sections provide details pertaining to main beacon registration table and those supporting tables that specifically configure the system at the application software level. The emphasis here is that the contents and/or schema of these tables are most pertinent at the compilation and underlying software development levels, and less so at the runtime level.

There are a number of other tables (data structures) that appear in the IBRD System Maintenance Manual as they are used to either configure the system at the runtime level or for runtime output such as logging. Several other tables in the database are simply discussed as appropriate in one or both documents in association with specific features or components of the IBRD System.

4.1 RegistrationDB406

4.1.1 Purpose

The RegistrationDB406 table is the main data table for the IBRD application. Its purpose is to store all the beacon registration data. In effect, this table is *the* beacon registration database, with all other tables providing supporting functions.

4.1.2 Table Layout

Field Name	Type	Bytes	Description	Req.[1]	Source[2]
BcnId15	char	15	Bits 26-85 of 406 MHz beacon message. Expressed as exactly 15 hexadecimal characters (0-9, A-F). Encoded position bits set to default values.	Yes	Data provider
CSTACNumber	varchar nullable	10	Cospas-Sarsat beacon type approval number		Data provider / beacon decode
BeaconRegType	tinyint	1	Type of beacon (See BeaconRegTypeCfg)		Beacon decode
BeaconType	varchar	32	Protocol used for beacon coding (See BeaconTypeCfg)		Beacon decode
BeaconCountryCode	smallint	2	Country Code from 406 Beacon Id		Beacon decode
BeaconActivationMethod	char nullable	4	Activation capability of beacon (See BeaconActivationMethodCfg)		Data provider

Field Name	Type	Bytes	Description	Req.[1]	Source[2]
BeaconManufacturer	varchar nullable	48	Name of manufacturer of beacon		Data provider
BeaconModel	varchar nullable	32	Model name of beacon		Data provider
SpecialStatus	varchar nullable	16	Indicates if the beacon is in-use, lost, stolen, sold, adrift, etc. (See SpecialStatusCfg)		data provider
SpecialStatusDate	datetime nullable	8	Associated Date/Time for change of beacon status		Database
SpecialStatusInfo	varchar nullable	255	Comments associated with Change in Status		data provider
PreviousSpecialStatus	varchar nullable	16	Store previous status of the beacon when a new status is provided (see above)		Database
BeaconHomingDevice	varchar nullable	10	Frequency or type of homing device. (See BeaconHomingDeviceCfg)		data provider / beacon decode
AdditionalBeaconData	varchar nullable	64	Any other information specific to the beacon that may be useful (e.g., manufacturers' serial number).		data provider
InitialDate	datetime	8	Date of original registration		Database
LastEditDate	datetime	8	Date that a field was last updated where the source was a data provider		Database
ConfirmPrintDate	datetime nullable	8	Date request for confirmation e-mail sent		Database
ConfirmationCompletedDate	datetime nullable	8	Date request for confirmation acknowledged by owner		Database
ConfirmationStatus	char nullable	4	Status of the "request for confirmation" process (See ConfirmationStatusCfg)		database / data provider
OwnerName	varchar	48	Full personal name, company name, or government agency name	Yes	data provider
Password	varchar	16	User password (Minimum of eight characters)	Yes	data provider
Address	varchar nullable	48	Owner's street or PO Box, postal code		data provider
City	varchar nullable	32	Owner's city		data provider

Field Name	Type	Bytes	Description	Req.[1]	Source[2]
Province	varchar nullable	32	Owner's province/state		data provider
MailCode	varchar nullable	16	Owner's mailing code		data provider
MailCountry	varchar nullable	60	Owner's country		data provider
EmailAddress	varchar nullable	48	Owner's E-mail address		data provider
Phone1Num	varchar	24	Owner's phone number	Yes	data provider
Phone1Type	char	4	Owner's phone type (See PhoneTypeCfg)	Yes	data provider
Phone2Num	varchar nullable	24	Owner's phone number		data provider
Phone2Type	char nullable	4	Owner's phone type (See PhoneTypeCfg)		data provider
Phone3Num	varchar nullable	24	Owner's phone number		data provider
Phone3Type	char nullable	4	Owner's phone type (See PhoneTypeCfg)		data provider
Phone4Num	varchar nullable	24	Owner's phone number		data provider
Phone4Type	char nullable	4	Owner's phone type (See PhoneTypeCfg)		data provider
PrimaryContactName	varchar	48	Name of primary emergency point of contact	Yes	data provider
PrimaryContactAddressLine1	varchar nullable	80	First line of address for primary emergency point of contact (e.g., street, apartment, etc.)		data provider
PrimaryContactAddressLine2	varchar nullable	80	Second line of address for primary emergency point of contact (e.g., city, province, country)		data provider
PrimaryPhone1Num	varchar	24	Primary emergency contact's phone number	Yes	data provider
PrimaryPhone1Type	char	4	Primary emergency contact's phone type (See PhoneTypeCfg)	Yes	data provider
PrimaryPhone2Num	varchar nullable	24	Primary emergency contact's phone number		data provider
PrimaryPhone2Type	char nullable	4	Primary emergency contact's phone type (See		data provider

Field Name	Type	Bytes	Description	Req.[1]	Source[2]
			PhoneTypeCfg)		
PrimaryPhone3Num	varchar nullable	24	Primary emergency contact's phone number		data provider
PrimaryPhone3Type	char nullable	4	Primary emergency contact's phone type (See PhoneTypeCfg)		data provider
PrimaryPhone4Num	varchar nullable	24	Primary emergency contact's phone number		data provider
PrimaryPhone4Type	char nullable	4	Primary emergency contact's phone type (See PhoneTypeCfg)		data provider
AlternateContactName	varchar nullable	48	Name of second emergency point of contact		data provider
AlternateContactAddressLine1	varchar nullable	80	First line of address for alternate emergency point of contact (e.g., street, apartment, etc.)		data provider
AlternateContactAddressLine2	varchar nullable	80	Second line of address for alternate emergency point of contact (e.g., city, province, country)		data provider
AlternatePhone1Num	varchar nullable	24	Second emergency contact's phone number		data provider
AlternatePhone1Type	char nullable	4	Second emergency contact's phone type (See PhoneTypeCfg)		data provider
AlternatePhone2Num	varchar nullable	24	Second emergency contact's phone number		data provider
AlternatePhone2Type	char nullable	4	Second emergency contact's phone type (See PhoneTypeCfg)		data provider
AlternatePhone3Num	varchar nullable	24	Second emergency contact's phone number		data provider
AlternatePhone3Type	char nullable	4	Second emergency contact's phone type (See PhoneTypeCfg)		data provider
AlternatePhone4Num	varchar nullable	24	Second emergency contact's phone number		data provider
AlternatePhone4Type	char nullable	4	Second emergency contact's phone type (See PhoneTypeCfg)		data provider
OperatorId	varchar	16	UserName/Logon Id (See Users Table) or the words "BEACON		Database

Field Name	Type	Bytes	Description	Req.[1]	Source[2]
			OWNER"		
BlockId	varchar nullable	16	User identification (or login name), specifically for National data providers only		Database
ChallengeQuestion	varchar nullable	64	Challenge question user selected for supporting re-instatement of password	Yes[3]	data provider
ChallengeResponse	varchar nullable	24	Challenge response user selected for challenge question for supporting re-instatement of password	Yes[3]	data provider
NumLogonFail	Tinyint nullable	1	Count of sequential logon failures for record. Used to deactivate record.		database
RecordStatus	char	1	Indicates whether or not record is active (See RecordStatusCfg)		database
AdditionalData	varchar nullable	255	Any other useful information about the beacon owner, the vehicle (e.g., tonnage, superstructure) and/or the specific usage of beacon.		data provider
VehicleType	varchar	48	Vehicle code for aircraft, vessel or personal use. (See EPIRBVehicleTypeCfg, ELTVehicleTypeCfg and PLBVehicleTypeCfg)	Yes	data provider
VehicleNationality	smallint nullable	2	MID country code for vessel flag State or aircraft nationality of registration.		data provider
VehicleName	varchar nullable	48	Name of vehicle or vessel, aircraft make and model	Yes[4]	data provider
MMSI	varchar nullable	9	Maritime Mobile Service Identity. Must be exactly 9 numerical characters (0-9) and extracted country code (first three characters) should match BeaconCountryCode	Yes[4,5]	data provider / beacon decode
Callsign	char nullable	10	Vessel radio call sign (See RadioCallSignCfg)	Yes[4,5]	data provider / beacon decode
VehicleRegistrationNumber	varchar nullable	16	IMO number or national vessel/ aircraft registration number	Yes[4]	data provider
Color	varchar	24	Color of vessel/aircraft		data

Field Name	Type	Bytes	Description	Req.[1]	Source[2]
	nullable				provider
Length	smallint nullable	2	Length of vessel/aircraft - Allowable range: 10 to 2000		data provider
Aircraft24BitAddress	char nullable	6	24-bit address of the aircraft, expressed as 6 hexadecimal characters		data provider / beacon decode
PeopleCapacity	smallint nullable	2	Vehicle capacity in numbers of people - Allowable range: 1 to 10000		data provider
VehicleCellularNum	varchar nullable	24	Cellular Telephone associated with Vehicle		data provider
PhoneInmarsat	varchar nullable	24	INMARSAT telephone number		data provider
RadioEquipment	varchar nullable	32	Radio equipment on board vessel/aircraft or person (See RadioEquipmentCfg)		data provider
SurvivalType1Num	smallint nullable	2	Number of survival equipment		data provider
SurvivalType1Desc	varchar nullable	64	Description of survival equipment		data provider
SurvivalType2Num	smallint nullable	2	Number of survival equipment		data provider
SurvivalType2Desc	varchar nullable	64	Description of survival equipment		data provider
AircraftOperatingAgency	varchar nullable	64	Aircraft operating agency designator and operator's serial number.		data provider

1. Abbreviation for "User Input Required".
2. Where the source indicates "data provider / beacon decode", the field will be automatically provided by the IBRD beacon decode software, whenever possible.
3. Mandatory for Data Providers (individual beacon owners) only, not mandatory when registration is controlled by a National Data Provider.
4. Not required for PLBs.
5. Not required for ELTs.

Primary Key – BcnId15
Index 1 – OwnerName
Index 2 – VehicleName
Index 3 – CallSign
Index 4 – VehicleRegistrationNumber
Index 5 – MMSI
Index 6 – VehicleType
Index 7 – LastEditDate
Index 8 – ConfirmPrintDate
Index 9 – BlockId

4.2 BeaconActivationMethodCfg

4.2.1 Purpose

The purpose of the BeaconActivationMethodCfg table is to store all the valid values for the beacon activation method for EPIRB beacons.

4.2.2 Table Layout

Field Name	Type	Bytes	Description
BeaconActivationMethod	Char	4	Values: CAT0 CAT1 CAT2
BeaconActivationMethodDescription	Varchar	40	Values: Category 0 - no data provided Category 1 - Automatic or Manual Category 2 - Manual only

Primary Key – BeaconActivationMethod

4.3 BeaconHomingDeviceCfg

4.3.1 Purpose

The purpose of the BeaconHomingDeviceCfg table is to store and provide a lookup for all the valid values for homing frequencies or devices.

4.3.2 Table Layout

Field Name	Type	Bytes	Description
BeaconHomingDeviceType	Char	1	Values: 1 S O N
BeaconHomingDeviceDescription	Varchar	16	Values: 121.5 MHz SART Other None

Primary Key – BeaconHomingDevice

4.4 BeaconRegTypeCfg

4.4.1 Purpose

The purpose of the BeaconRegTypeCfg table is to store all the valid values for the registration type.

4.4.2 Table Layout

Field Name	Type	Bytes	Description
BeaconRegType	tinyint	1	Valid Values: 0 1 2 Additional Unused values: 3 7 8 9
BeaconRegName	varchar	16	Valid Values: EPIRB ELT PLB Additional Unused values: SSAS Test Orbitography National Use

Primary Key – BeaconRegType
Index (unique) – BeaconRegName

4.5 BeaconTypeCfg

4.5.1 Purpose

The purpose of the BeaconTypeCfg table is to store all the values for many different types of beacons, based on protocol.

4.5.2 Table Layout

Field Name	Type	Bytes	Description
BeaconType	varchar	32	Values: ELT 24 BIT ADDRESS (STD) ELT A/C OPERATOR (STD)

Field Name	Type	Bytes	Description
			ELT AVIATION USE ELT SERIAL (NATIONAL) ELT SERIAL (STANDARD) ELT SERIAL A/C 24BIT ADDRESS ELT SERIAL A/C OPERATOR ELT SERIAL AVIATION EPIRB MARITIME USER EPIRB MMSI (STANDARD) EPIRB RADIO CALL SIGN EPIRB SERIAL (NATIONAL) EPIRB SERIAL (STANDARD) EPIRB SERIAL CATEGORY I EPIRB SERIAL CATEGORY II NATIONAL USER ORBITOGRAPHY ORBITOGRAPHY (RESERVED) PLB SERIAL PLB SERIAL (NATIONAL) PLB SERIAL (STANDARD) TEST TEST (NATIONAL) TEST SERIAL (STANDARD)

Primary Key – BeaconType

4.6 ConfirmationStatusCfg

4.6.1 Purpose

The purpose of the ConfirmationStatusCfg table is to store all the valid values for the status types of the “request for confirmation” process.

4.6.2 Table Layout

Field Name	Type	Bytes	Description
ConfirmationStatus	char	4	Values: SENT – means that a confirmation request has been sent out UDEL – means that a confirmation request has been marked as undeliverable by database administrator CFRM – means that a confirmation request has been Acknowledged on line (no changes were needed)

Field Name	Type	Bytes	Description
			CHGE – default value in effect which means that changes have been made and hence confirmation is implied
ConfirmationStatusDescription	varchar	50	Not Used – appropriate translations are pulled from formlabel properties files

Primary Key – ConfirmationStatus

4.7 PhoneTypeCfg

4.7.1 Purpose

The purpose of the PhoneTypeCfg table is to store and provide a lookup for all the valid values for the types of telephone numbers for a beacon registration.

4.7.2 Table Layout

Field Name	Type	Bytes	Description
PhoneType	char	4	Values: HOME WORK CELL FAX OTHR
PhoneTypeDescription	varchar	10	Not Used – appropriate translations are pulled from formlabel properties files

Primary Key – PhoneType

4.8 RadioCallSignCfg

4.8.1 Purpose

The purpose of the RadioCallSignCfg table is to store all the valid range values for the radio call signs for each organization (country name). Each organization (country name) is assigned a range of Call Sign Series. Based on the beacon identification code's encoded country code a country name is determined from the MidInfoCfg table.

Validation Logic: If there is not a match between the encoded country code (i.e., the country name it maps to) and the call sign's associated country name, a warning message will be generated.

4.8.2 Table Layout

Field Name	Type	Bytes	Description
CallSignFirst	char	3	First Call Sign Series associated with an Organization (Country Name)
CallSignLast	char	3	Last Call Sign Series associated with an Organization (Country Name)
OrgName	varchar	16	Owners' Organization (Country Name) associated with Call Sign Series First and Call Sign Series Last

Primary Key – CallSignFirst, CallSignLast, OrgName

4.9 RadioEquipmentCfg

4.9.1 Purpose

The purpose of the RadioEquipmentCfg table is to store all the valid values for the types of radio equipment on board a vessel or aircraft. This table provides a lookup of valid radio equipment types.

When multiple radio equipment options (check boxes) are selected, the data to store in the Beacon Registration table stores these multiple selections separated by a commas. Under the option "Other" the user is permitted to type in their own radio equipment type. If the user selects the option "Other", and still leaves the text blank, then the actual text "Other" will be inserted.

4.9.1 Table Layout

Field Name	Type	Bytes	Description
RadioEquipment	varchar	5	Values: VHF HF MF SSB Other

Primary Key – RadioEquipment

4.10 RecordStatusCfg

4.10.1 Purpose

The purpose of the RecordStatusCfg table is to store all the valid values for the record status. This table provides a lookup of valid record statuses to

capture whether a beacon record is active or deactivated (due to lockout for password entry failures).

4.10.2 Table Layout

Field Name	Type	Bytes	Description
RecordStatus	char	1	Values: A D
RecordStatusDescription	varchar	20	Values: Active Deactivated

Primary Key – RecordStatus
Index 1 (unique) – RecordStatusDescription

4.11 RolesCfg

4.11.1 Purpose

The purpose of the RolesCfg table is to store access role information to determine the type of user. This information is used to determine the ability of users to perform different functions in the application.

4.11.2 Table Layout

Field Name	Type	Bytes	Description
RoleId	int	4	Role Identifier Values: 1 2 3 4
RoleName	varchar	30	Role Name Values: BLOCK USER (National Data Provider) SHIP SURVEYOR USER SYSTEM MANAGER (Database Admin.) SAR USER (SAR Services)

Primary Key – RoleId
Index (unique) – RoleName

4.12 SpecialStatusCfg

4.12.1 Purpose

The purpose of the SpecialStatusCfg table is to store all the valid values for the changes in status regarding the beacon in the database.

4.12.2 Table Layout

Field Name	Type	Bytes	Description
SpecialStatus	varchar	16	Values: LOST STOLEN SOLD REPLACED DESTROYED OUTOFSERVICE (NULL = Normal Status)
SpecialStatusDescription	varchar	30	Not Used – appropriate translations are pulled from formlabel properties files

Primary Key – SpecialStatus

4.13 ELTVehicleTypeCfg

4.13.1 Purpose

The purpose of the ELTVehicleTypeCfg table is to store and provide a lookup for all the valid values for the vehicle type of aircrafts associated with ELT beacons.

Under the option "Other" the user is permitted to type in their own radio equipment type. If the user selects the option "Other", and still leaves the text blank, then the actual text "Other" will be inserted.

4.13.2 Table Layout

Field Name	Type	Bytes	Description
ELTVehicleType	varchar	48	Values: Single-engine Propeller Single-engine Jet Multi-engine Propeller Multi-engine Jet Helicopter Other

Primary Key – ELTVehicleType

4.14 EPIRBVehicleTypeCfg

4.14.1 Purpose

The purpose of the EPIRBVehicleTypeCfg table is to store and provide a lookup for all the valid values for the vehicle type of vessels associated with EPIRB beacons.

Under the option “Other” (two cases, POWER or NON-POWER) the user is permitted to type in their own radio equipment type. If the user selects the option “Other”, and still leaves the text blank, then the actual text “Other” will be inserted.

4.14.2 Table Layout

Field Name	Type	Bytes	Description
EPIRBVehicleType	varchar	48	Values: SAIL (prompts an entry for number of Masts. Text in database appears as "SAIL nn Masts") POWER Fishing POWER Tug POWER Cargo POWER Tanker POWER Pleasure Craft POWER “Other” NON-POWER Life Boat NON-POWER Life Raft NON-POWER “Other”

Primary Key – EPIRBVehicleType

4.15 PLBVehicleTypeCfg

4.15.1 Purpose

The purpose of the PLBVehicleTypeCfg table is to store and provide a lookup for all the valid values for the vehicle type associated with the specific use of PLB beacons.

Under the option “Other” the user is permitted to type in their own radio equipment type. If the user selects the option “Other”, and still leaves the text blank, then the actual text “Other” will be inserted.

4.15.2 Table Layout

Field Name	Type	Bytes	Description
PLBVehicleType	varchar	48	Values: Land Vehicle Boat Aircraft None Other

Primary Key – PLBVehicleType

4.16 UsageMoreInfoCfg

4.16.1 Purpose

The purpose of the UsageMoreInfoCfg table is to store and provide a lookup for all the valid values for additional usage information pertaining to PLB beacons.

Under the option "Other" the user is permitted to type in their own radio equipment type. If the user selects the option "Other", and still leaves the text blank, then the actual text "Other" will be inserted.

4.16.2 Table Layout

Field Name	Type	Bytes	Description
UsageMoreInfo	varchar	24	Values: Fishing Hunting Hiking Other

Primary Key – UsageMoreInfo

4.17 PasswordChallengeCfg

4.17.1 Purpose

The purpose of the PasswordChallengeCfg is to hold the values for challenges questions that are used by the system to provide forgotten passwords to beacon owners. There are actually four tables to support the multi-lingual interface each containing the name of the language, English, French, Russian and Spanish (e.g., PasswordChallengeEnglishCfg).

4.17.2 Table Layout

Field Name	Type	Bytes	Description
ChallengeQuestion	varchar	64	Values – Given here in English – other languages in each table as stated above: What is the name of your favourite movie? What is the name of your favourite teacher? What is the name of your pet? What is your favourite sports team? What is your mother's maiden name? What was the name of your high school? What was your childhood hero?

Primary Key – ChallengeQuestion

4.18 MailCountryCfg

4.18.1 Purpose

The purpose of the MailCountryCfg table is to store and provide a lookup for all the valid values for country names stored in the beacon owner's mailing address. The interface as implemented at the time of Acceptance Testing only provides this listing (or pull-down menu) in the English Language.

4.18.2 Table Layout

Field Name	Type	Bytes	Description
MailCountry	varchar	60	Values: (Too lengthy for practical listing here)

Primary Key – MailCountry

- END OF SECTION 4 -

5. J2EE BACKGROUND

The Main IBRD Application is built using Java 2 Enterprise Edition (J2EE) technology. As such, some general background on this paradigm is provided.

5.1 EAR Structure

A J2EE application is packaged as a portable deployment unit called an enterprise archive (EAR) file. An EAR file is standard Java Archive File (JAR) file with an `.ear` extension. An EAR file contains:

- One or more J2EE modules
- One J2EE application deployment descriptor

Creation of a J2EE application is a two-step process. First, application software developers create various client modules. Second, the application assembler packages these modules together to create a J2EE application module that is ready for deployment. In the case of the IBRD System, these modules consist of one Web archive (WAR) file and several JAR files, two custom and several open source third-party modules. Section 5.2, just below, provides further information on WAR files. The IBRD EAR file also contains two more files, a deployment descriptor file (application.xml) and a manifest (manifest.mf) file, which clearly help to define the deployment structure. More details on the IBRD deployment structure are provided in Section 6.1 below.

All J2EE modules are independently deployable units. This enables software developers to create independent units of functionality without having to implement full-scale applications. To assemble an application, an application assembler resolves dependencies between components by creating links in the corresponding modules' deployment descriptors. Each component may have dependencies on other components within the same archive, on components in different archives, or both. All such dependencies must be resolved before deployment.

5.2 WAR Structure

A Web module is packaged and deployed as a Web archive (WAR) file, a JAR file with a .war extension. It is the smallest deployable and usable unit of Web resources. A Web module may contain:

- Java class files for the servlets and the classes that they depend on, optionally packaged as a library JAR file
- JSP pages and their helper Java classes
- Static documents (for example, HTML, images files, etc)
- Applets and their class files
- Web deployment descriptor

Unlike other deployment unit types, a WAR file usually cannot be loaded by a ClassLoader, because its internal folder structure differs from that of a loadable JAR file. Like other module types, a WAR file may be deployed independently as a Web application or packaged in an EAR file and deployed as a J2EE application.

The Web module is the smallest indivisible unit of Web resources that an application component provider supplies to the application assembler. Understanding how Web application components map into a server address space requires an understanding of the structure of a request Uniform Resource Identifier (URI). The URI representing a request to a Web component is called a request path. After the protocol and hostname, a request URI has the following components:

- The context path locates the Web application in the Web server's namespace at deployment time. It can be thought of as the path to the "root folder" of a Web application (called the context root), relative to the root of the Web server namespace. A context path is always either empty (meaning that the root of the Web application is the root of the Web server namespace) or it both begins with a slash and does not end with one.
- The servlet path is the part of the URI that matched the servlet mapping for the request. It appears directly after the context path and never begins with a slash.
- The path info is any part of the request URI that is not part of the context path or the servlet path that follows the server path but precedes the query string. The HTTP GET query string, for example, typically appears as path info. Path info may be empty.

Given the following request example: https://localhost/IBRD/Dispatch?page=Start; the context path is “/IBRD/”, the servlet path is “/Dispatch?”, and the path info is “page=Start”. Except for URL encoding details, a valid request URI is always a context path, followed by a servlet path, followed by path info.

The Java Servlet specification defines a mandatory folder structure for a Web application deployment unit. The Web application folder structure applies to the internal structure of a WAR file. The Java Servlet specification recommends, but does not require, that this same structure also be used as a runtime representation. Figure 5.1 below shows this structure graphically.

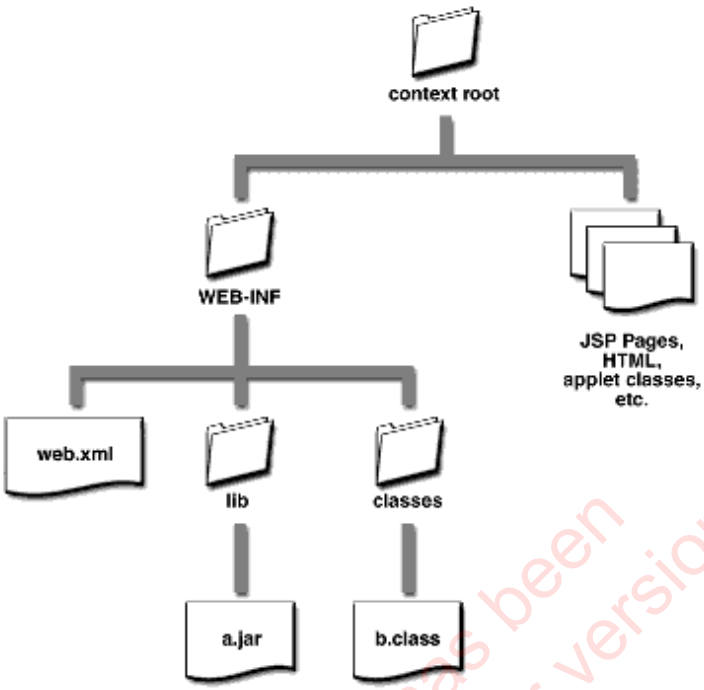

Figure 5.1 Web Application Folder Structure

The root folder of the Web application is the context root, which is mapped to the context path at deployment time. The context root contains the application's JSP pages, content, graphics, applet classes, and other files that the application serves to clients. These files are shown on the right in the above figure. Also under the context root is the WEB-INF folder, which contains files that are not intended to be served to clients. The WEB-INF folder has a specific structure, and has the following contents:

- The deployment descriptor file, called web.xml
- A folder called lib, which may contain JAR files that will automatically be added to application components' classpath at runtime. Third-party libraries often reside in this folder.
- A folder called classes, which contains any classes needed by the application that are not in a JAR file. Such classes must be organized in folders by package, as usual.

The IBRD System follows this structure in general terms. For more details on the IBRD WAR file see Section 6.1 below.

- END OF SECTION 5 -

Page left blank

6. MAIN APPLICATION STRUCTURE

6.1 IBRD Deployment Structure

The Main IBRD Application is deployed as two archive files. The core application is completely supported by a single IBRD Enterprise Archive file which is named, IBRD.ear. All required components are stored in this archive.

Additionally, one JAR file which is contained within the EAR file is also deployed separately. Deployed in this fashion, a file named IBRD_util.jar file provides a second and independent point of entry for the "Request for Confirmation" process which is run as a "stand-alone" scheduled task. The IBRD_util.jar file is discussed below in Section 6.1.3, and the "Request for Confirmation" process is discussed further in Section 8.12.

6.1.1 Overall Structure of EAR File

The IBRD EAR consists of the following files:

- activation.jar
- jacob.jar
- mail.jar
- xerces.jar
- IBRD.war
- IBRD_util.jar
- IBRDEJB.jar
- application.xml
- manifest.mf

The activation.jar, mail.jar, jacob.jar and xerces.jar are all JAR files from third party open source providers. The IBRD.war, IBRD_util.jar and IBRDEJB.jar contain custom software developed specifically for this application although some of the Enterprise Java Beans (EJB) would qualify as third party open source code, customized as appropriate.

The application.xml file is the XML deployment descriptor for the EAR file and is located in the META-INF subfolder of the application archive. The application.xml file contains information about the modules that comprise the EAR file. The application.xml file is discussed in more detail in Section 8.1.

The Manifest.mf is a file which commonly consists of a list of the files present within the archive itself. However, not all files in the archive need to be listed in the manifest and although a manifest file is accordingly included in the EAR file and each other archive file as well, it is largely unused providing only a preliminary section containing, at minimum, this standard's version number and the list of third party JAR files as appropriate for the archive.

6.1.2 Third Party JAR Components

The activation.jar and mail.jar are implementation packages of the JavaMail API, a set of APIs that model a mail system. The activation.jar contains Java class files that are a part of the JavaBeans Activation Framework (JAF). The JavaMail API uses the JAF for data content handling of the email. The mail.jar contains Java class files that implement the core JavaMail packages which provide facilities for reading and sending email and supports multiple protocols and service providers including SMTP, IMAP and POP3.

The jacob.jar contains the Java class files that support JNI interface to perform native calls to COM and Win-32 libraries (such as the Beacon Decode DLL). The xerces.jar provides a parser for the eXtensible Markup Language (XML) and contains the Xerces Java Parser that comes packaged with API documentation for SAX and DOM, the two most common interfaces for programming XML. The Xerces Java Parser is used for parsing some of the XML properties files that contain customizable and deployment information for the main IBRD application.

6.1.3 WAR File

The IBRD Web Archive file is called IBRD.war. The IBRD WAR consists of multiples files and folders that contain all the files for what is considered the "web application". Specifically, this file focuses on all of the mechanisms that pertain to the front-end or user interface with the other two IBRD JAR files providing supporting functions or back-end capabilities.

As described in Section 5.2 above, several subfolders are found within the WAR file, which contain the various elements that combine to provide the "web application". This structure is briefly outlined here with further folder specific details in provided in Section 6.2 just below and module to module mappings provided in detail in Section 7.

The user interface itself, is largely provided by Java Server Pages (JSP). These JSPs are dynamic files that create the application's HTML which in turn is presented to end users. These files are found in the jsp\ subfolder of the archive. The resources\ subfolder contains the Properties files that provide various on screen text in the multiple languages of the IBRD interface. Each with additional subfolders for multi-lingual support, the help\ folder contains the applicable HTML, the images\ folder contains various GIF and JPEG formats and the js\ folder contains Java Scripts. Finally the css\ folder contains the single cascading style sheet for the main IBRD application. There is also a meta-inf subfolder which holds the single manifest.mf file.

Again as described in Section 5.2 above, all code served up by the Application Server (however not visible to the client) is stored in the WEB-INF folder. The subfolders under the \WEB-INF\classes\ folder contain all the class files directly needed by the JSPs which includes the Tag Library classes (subfolder taglib\form\) and various user interface support including Servlets (subfolder ui\). Several other software configuration/control files are stored in the WEB-INF

folder. The web application deployment descriptor, named web.xml, is here as well as several other supporting software configuration/control XML files: dbreg.xml, acl.xml, jrun-resources.xml and log4j.xml. Finally, the WEB-INF folder in the WAR file contains two custom tag library descriptor files, noaa-form.tld and noaa-framework.tld. All of these files configuration/control are further discussed in associated subsections of Section 8

6.1.4 IBRD_util JAR File

All the underlying or complex or back-end support for the web application (IBRD WAR file), other than EJB components, are found in the IBRD_util.jar archive file. Other than the set of resource Properties files for multi-lingual support, all files in this Java archive are Java class files. The subfolders for classes include: complex\, complex\tools\, exception\, framework\, log\ and reporting\. Descriptions for these subfolders can be found below in Section 6.2. There is also a meta-inf which holds the single default format manifest.mf file. As noted above, this one archive is deployed separately as well as within the IBRD EAR file to support the independently executed "Request for Confirmation" process.

6.1.5 EJB JAR File

The IBRDEJB.jar contains all of the Java class files for the EJB components of the main IBRD application. The subfolders for classes include: complex\ejb\, framework\ejb\, log\ejb\ and reporting\ejb\. Descriptions for these subfolders can be found below in Section 6.2. There is also a meta-inf subfolder which holds a minimal information manifest.mf file as well as a deployment descriptor for all the EJBs, named ejb-jar.xml. This deployment descriptor is further discussed in Section 8.4.

6.2 Software Development/Deployment Folder Structure

In the development environment, the IBRD software is laid out in a set of folders similar by definition to the deployment structure discussed above. Before proceeding to Section 7 which details the various mappings of between front-end interface modules (JSP, HTML etc.) and the associated underlying support modules (Servlets, Java Beans, etc.) it is useful to discuss the overall layout and categorization of modules and folders. For more information on software development see Sections 9.

Figure 6-1 below shows the major folders and subfolder within the software development folder. Brief descriptions of the related contents of the various subfolders follow. Relative to Java "package" specifications it is useful to note that the folder name "cs\ibrd\" translates to the same package prefix "cs.ibrd." for all Java classes in the main IBRD application. Several cases may be noted under the "web" subfolder where there are four additional subfolders with the names, English, French, Russian and Spanish. In the IBRD System, the facilities under the "Login" point of entry are only provided only in English where as those found under the "Data Provider Welcome Page" (index.jsp) are multi-lingual. In each of these situations, the main subfolder contains files that pertains predominately users who use the "Login" (Login.jsp) page (SAR Users, Ship Surveyors and Database Administrators) and the four

language subfolders clearly hold the analogous files for the multi-lingual portion of the user interface.

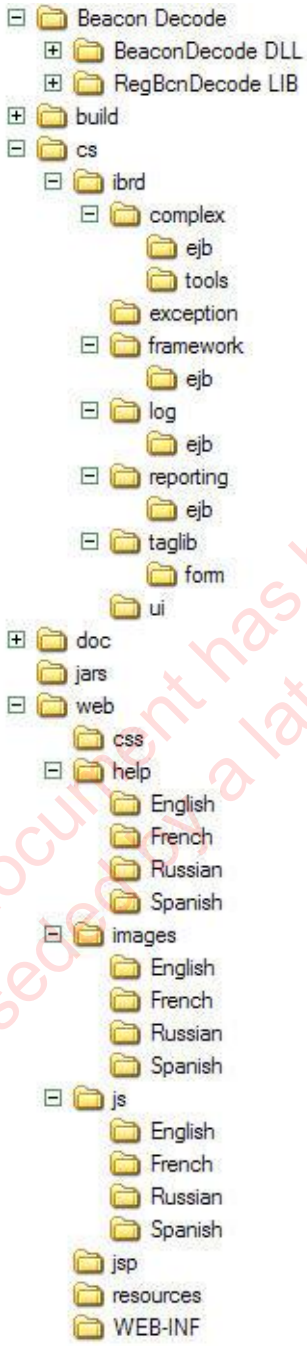

Figure 6.1– Software Folder Structure

A brief description of the folder and subfolder contents is given as follows for the structure shown above in Figure 6.1:

Folders for the WAR File and Front-End Interface

- *web\css* – This folder contains the IBRD cascading style sheet used for Font declarations, color mappings and other layout settings for JSP and HTML objects.
- *web\help* – This folder and its subfolders contains the HTML files for all help pages that are accessed within the application.

- *web\images* – This folder contains the .GIF and .JPG files that have the images for the application (e.g., background, logos, interface buttons).
- *web\js* – This folder contains all of the JavaScript files used to support JSP and HTML objects.
- *web\jsp* – This folder contains all of the Java Server Pages (JSPs) to support the IBRD application. For all the web pages of the application, JSPs are used to build the HTML for dynamic data display and processing.
- *web\resources* – This folder contains the Properties files for the application that contain on screen labeling and errors messages. There are two sets of four language files, one set for form entry labels and the other set for error messages.
- *web\WEB-INF* – This folder contains the application data that is not accessible from browser access and is only available to the application server. This folder contains XML files which store customizable data used by the application. Areas include access control and authentication (acl.xml), beacon table data declarations (dbReg.xml), logging set up parameters (log4j.xml) and web application environment settings (web.xml). Additionally, this folder contains tag library files (TLD files) to be used with the tag library Java code for setting of tag specifications (noaa-form.tld and noaa-framework.tld). For more information on each these files see the associated subsections of Section 8.
- *cs\ibrd\taglib\form* – This folder contains the Java classes used for the custom tag libraries which dynamically create HTML for many of the form objects. These custom tags include Drop Down lists, Pick lists, Radio lists, Text Area inputs and Text Field inputs among others.
- *cs\ibrd\ui* – This folder contains the Java classes for the all of the application Java Servlets, the Java Beans and supporting classes for data entry and data validation. Servlet classes usually contain Servlet in the class name, the Bean classes usually contain Bean in the class name and the Field Validation classes usually contain FV in the class name.

Folders for Back-End Support: IBRD_util.jar and IBRDEJB.jar

- *cs\ibrd\complex* – These folders contain the Java classes (includes EJBs and tools) for the Complex and Background Processing components that are used for Beacon Access (Code Values for Configuration tables, View and Update), Beacon Query (Search\Filter), Email Transmit Subsystem and Request for Confirmation process.
- *cs\ibrd\exception* – This folder contains the Java classes for handling various IBRD application exceptions (errors in "object oriented" / Java terminology).
- *cs\ibrd\framework* – These folders contain the Java classes (includes EJBs) for the components in the "framework" or internal workings of the application. These components are the shared code for Configuration data access, the Login mechanism for all types of users, and the base Servlets that are used as the parent class for inheritance for all the application Java Servlets. This folder also contains code that is shared between the User Interface as well as the Complex and Background Processing components such as for application constants and utilities for accessing the EJBs.
- *cs\ibrd\log* – These folders contain the Java classes (includes EJBs) for the logging mechanism within the application. These classes are built to support the open source Log4J logging package that has been customized for IBRD application needs. This logging supports: Database Change logging, Email logging, Query Access logging, User Access logging, record Transaction logging, and error/exception logging.

- *cs\ibrd\reporting* – These folders contain the Java classes (includes EJBs) to support the Reporting functionality for both the event tracking and report presentation processes.

Miscellaneous Folders

- *BeaconDecode* – This folder contains two subfolders. One subfolder (RegBcnDecode LIB\) contains the core C++ code that provides the Beacon Decode functionality itself. The other subfolder (BeaconDecode DLL\) contains C++ code that provides the JNI interface and results in the actual DLL (IBRD_ BeaconDecode.dll) that is called from the Java code of the main IBRD application.
- *jars* – This folder contains several third-party open source Java Archive files (JARs) that are needed for the IBRD application. Included are support for: the mail process (application.jar and mail.jar), XML processing (xerces.jar), Beacon Decode library interface (jacob.jar), Logging (log4j-1.2.4.jar) and basic J2EE core support (J2EE.jar).
- *build* – This is a temporary storage folder for the results of a "build" operation for the main IBRD application. Specifically, all the custom JAR (IBRD_util.jar and IBRDEJB.jar), WAR (IBRD.war) and eventually the IBRD EAR (IBRD.ear) files are created in this folder as well as all the individual class files in appropriately structured subfolders.
- *doc* – This is a temporary storage folder for documentation automatically generated by JavaDoc.

- END OF SECTION 6 -

7. MAIN APPLICATION MODULE MAPPINGS

The following table maps the front-end pages and general functions in the IBRD system to the underlying JSP, package subfolder and help page as applicable associated with the page or function. The table provides a generalized mapping of pages and functions relative to the resources that they use. Other views of the software modules and their interdependencies can be generated with industry standard tools such as JavaDoc.

It is useful to note that where "National Data Provider" support is provided, the underlying code will usually use "block" in the class (or file) name. This is due to the origin of the application software where a similar functionality already existing for owner of a "block" of beacons. The concept of a "block" (e.g., many beacons owned by one business) was extended and modified to provide the "National Data Provider" functionality. It may also be useful to note that "Data Providers" are analogous to "Beacon Owners" with similar effect on the historical names of class (and hence files). Finally, usage of "NOAA" in a file name is analogous to support for Database Administrators and/or the general system, once again coming from the origins of the software as a USA based system housed at NOAA facilities.

Wherever pages or functions are virtually the same under different user scenarios, the page is only mentioned once. For example, all users who have the capability to update an existing record (Data Providers, National Data Providers and Database Administrators) will see essentially the same page and hence this functionality only appears once in the following table. However, relative to the various user types, the interface differs somewhat more significantly when viewing a beacon, resulting in perhaps a different help page etc. and hence these functions appear in multiple rows as applicable.

Table 7.1 Module Mappings

Page / Function	JSP (*.jsp)	Package	Class / Servlet (*.java)	Help Page (*.htm)
Data Providers				
Language Selection	index	ui	LanguageServlet	
Agreement / Disclaimer	Agreement			
Data Provider Welcome	DataProviderWelcome			
Data Provider Login	BO_login	framework	LoginServlet	BO_login_help
National Data Provider Login	BU_login	framework	LoginServlet	BU_login_help
Beacon password reminder	BO_request_pwd password_mailed	framework	RequestPassword	BO_request_pwd_help
National Data Provider password reminder	BU_request_pwd, password_mailed	framework	RequestPassword	BU_request_pwd_help
Data Provider Home page	BeaconForm	ui	BeaconFormServlet	owner_beacon_view_help
Registration new beacon	register_new_beacon	framework	LoginServlet.java	register_new_beacon_help
Update existing beacon	BeaconForm	ui	BeaconFormServlet BeaconUpdateServlet	owner_beacon_update_help beacon_registration_fields_help
Acknowledge Confirmation Request	ConfirmStatus	ui	ConfirmStatusServlet	confirm_status_help
Change beacon status	BeaconStatus	ui	BeaconStatusServlet	lost_stolen_help
Change beacon password	BeaconPassword	ui	BeaconPasswordServlet	changepassword_help
National Data Provider Home page	BlockHome	ui	BlockHomeServlet	block_beacon_owner_homepage_help
National Data Provider View beacon	BeaconForm	ui	BeaconFormServlet	block_beacon_view_help
Upload Records	BlockHome GetBulkUploadFile	ui	BlockHomeServlet GetBulkUploadFileServlet	
Assign beacon to National List	AssignBeacon	ui	AssignBeaconServlet	block_add_beacon_help
Remove beacon from National List		ui	BlockHomeServlet	
Unsupported country code	UnsupportedCC			

Page / Function	JSP (*.jsp)	Package	Class / Servlet (*.java)	Help Page (*.htm)
SAR Users and Ship Surveyors				
User Login	Login	framework	LoginServlet	login_help
Agreement / Disclaimer	SarAgreement			
Search options and results	NOAAHome	ui	SearchServlet	sar_search_beacons_help
View registered beacon	BeaconForm	ui	BeaconFormServlet BeaconUpdateServlet	sar_beacon_view_help
Database Administrator				
Search options and results	NOAAHome	ui	SearchServlet	search_beacons_help
Manage beacon account	ManageBeacon	ui	ManageBeaconServlet	manage_beacon_help
Access National Beacons	AccessBlock	ui	AccessBlockServlet	
Report Options	ReportOptions	ui	ReportOptionsServlet	report_help
Report	BeaconReport	ui		
Administration Find Accounts	AccountSearch	ui	AccountSearchServlet	admin_help
User Accounts Listing	RegisteredAccounts	ui	AccountSearchServlet	queried_accounts_help
Add new user account	add_account	ui	AddAccountServlet	add_account_help
Update user account	add_account	ui	AccountManagerServlet	update_account_help
Delete user account	RegisteredAccounts	ui	AccountSearchServlet	
View Points of Contact	CountryCodes	ui	CountryCodesServlet	
Footers				
Log Out			HomeServlet	
Home	footer_redirect		HomeServlet	
Help				help general_help
Feedback survey	survey		FeedbackSurveyServlet	survey_help
Contact us	ContactUs			
Privacy policy				privacy

Page / Function	JSP (*.jsp)	Package	Class / Servlet (*.java)	Help Page (*.htm)
Complex / Background				
Authentication		framework framework.ejb	NOAAServlet LoginServlet RequestPasswordServlet	
Access Control		framework framework.ejb	NOAAServlet ValidationServlet DispatchServlet	
Session Management		framework framework.ejb	LoginServlet DispatchServlet	
Exception Handling		exception	(entire folder)	
Request for Confirmation		complex	TwoYearCfmRequest TwoYearCfmRequestDAO TwoYearCfmRequestServlet	
Logging		log log.ejb	(entire folder)	
Reporting		reporting reporting.ejb	(entire folder)	

- END OF SECTION 7 -

8. INTERNAL STRUCTURES AND SUPPORTING ELEMENTS

The following subsections discuss major software configuration structures as well as some significant supporting software elements for the IBRD System. Various files discussed below contain information that provides for internal software customization of IBRD application. The information held in these elements encompass a wide range of settings that include: module names, data source names for accessing databases, web application names for servlets, access control for pages based upon user roles, email templates and defaults, and application specific parameters. There are also many SQL configuration tables that control the behavior of the IBRD system. Most of these tables are discussed above in Section 4. The purpose of all these files and tables is to configure the application at "compile" (or build) time.

Within the IBRD application software, eXtensible Markup Language (XML) files provide data or information to the application in a text based format that uses tags analogous to those found in Hyper Text Markup Language (HTML) files. These XML tags have starting and ending delimiters of the form "<tagname></tagname>" and are often nested. Information needed by the software is obtained by parsing these files. For further details regarding XML, appropriate related documentation should be consulted. The XML files that are used by the IBRD application are discussed accordingly in the following subsections.

Much like XML files, "Properties" files are text files that also provide information to the IBRD application. In this case the format is essentially limited to individual lines containing a <name/key>=<value> format. Again information needed by the software is obtained by parsing these files which are discussed below.

8.1 Application Deployment XML Descriptor

This file is named "application.xml" and is dynamically generated during a system build from the file named "application-template.xml" file which is found in the root of the software development folder. The "application.xml" file is the deployment descriptor for the EAR file, and during a build is stored in the "META-INF" subfolder of the application archive itself. The "application.xml" file contains information about the modules that are found in the EAR file.

The "application.xml" file must begin with the following DOCTYPE declaration. It should be noted that all XML files used by the IBRD application contain this same declaration:

```
<!DOCTYPE application PUBLIC "-//Sun Microsystems,
Inc.//DTD J2EE Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/application_1_2.dtd">
```

Example "application.xml" file (initial deployment for Acceptance Testing):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE application PUBLIC "-//Sun Microsystems, Inc.//DTD J2EE Application 1.2//EN"
"http://java.sun.com/j2ee/dtds/application_1_2.dtd">
<application>
    <icon><small-icon>smallIcon.gif</small-icon></icon>
    <display-name>IBRD</display-name>
```

```
    <description>J2EE Application</description>
    <module>
        <ejb>IBRDEJB.jar</ejb>
    </module>
    <module>
       <web>
           <web-uri>IBRD-war</web-uri>
           <context-root>IBRD</context-root>
       </web>
    </module>
</application>
```

The following tags in the application.xml file are noted:

- <application></application> - contains tags about the application to be deployed
- <display-name></display-name> - contains the name of the application that is displayed in the application server console
- <module></module> - contains tags about the different modules that comprise the application (in the example above, there are EJB and Web modules).
- <ejb></ejb> - contains the name of EJB JARs which contain the Enterprise Java Beans for the application (for business logic and data access)
- <web></web> - contains the name of the Web application WAR and the context root for the URI
- <context-root></context-root> - defines the context root for the application. Specifically, in the URL used for the IBRD system (e.g., www.cospas-sarsat.org\ibrd\) this is how the "\ibrd\" portion is properly mapped to the application software.

8.2 JRun Resource XML Descriptor

This XML file specifies all of the JRun resources for a JRun server and contains configuration information for all J2EE Resource Factories: JDBC, JMS, Mail, and URL. The "jrun-resources.xml" file can be used by JRun to configure many application server resources as J2EE applications typically share certain resources. This lets the application server ensure consistent availability, naming and resource pooling. This file is located in the "\web\resources" subfolder.

For the purposes of the IBRD application software, changes are not generally necessary or expected to be required for this file. Rather, JRun resources (e.g., connection to SQL Server database) for the IBRD application are configured using the JRun Management Console (see Section 11 regarding IBRD System Installation).

8.3 Web Application Deployment XML Descriptor

The "web.xml" file is the Web Application deployment descriptor for the application. This XML file describes the servlets and other components that make up the application, along with any initialization parameters and container-managed security constraints that the server is to enforce. The "web.xml" file can is found in "\web\WEB-INF" subfolder and during a build is stored in the "web-inf" subfolder of the IBRD WAR file.

Specifically, the "web.xml" file identifies all of the servlets and any provides information needed that by the servlet containers that comprise the web application portion of the IBRD system. This file is used to register the servlets to the JRun Servlet Container and may

include the servlet name, the servlet class, any parameters the servlet accepts as well as startup requirements.

This file also contains several other system critical element definitions: session configuration information for the application to configure how long a session should last (or be timed out if no activity occurs for that time period); the "welcome" or start up page for the application; and information about the tag libraries used in the servlets (See Section 8.8 below for more information about tag libraries).

Example extracts from the "web.xml" file (vertical dots indicate omitted portions):

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>
    <display-name>NOAA Framework Test</display-name>

   <servlet>
       <servlet-name>LoginServlet</servlet-name>
        <servlet-class>cs.ibrd.framework.LoginServlet</servlet-class>
   </servlet>

   <servlet>
       <servlet-name>DispatchServlet</servlet-name>
       <servlet-class>cs.ibrd.framework.DispatchServlet</servlet-class>
       <!-- the following init-param sets the config filename for Log4J -->
    <init-param>
          <param-name>log4j-init-file</param-name>
          <param-value>/WEB-INF/log4j.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet>
       <servlet-name>RequestPasswordServlet</servlet-name>
       <servlet-class>cs.ibrd.framework.RequestPasswordServlet</servlet-class>
   </servlet>

.
.
.
.
.

   <servlet>
       <servlet-name>GetBulkUploadFileServlet</servlet-name>
       <servlet-class>cs.ibrd.ui.GetBulkUploadFileServlet</servlet-class>
   </servlet>

    <servlet-mapping>
        <servlet-name>LoginServlet</servlet-name>
        <url-pattern>/Login</url-pattern>
    </servlet-mapping>

.
.
.
.
.
    <servlet-mapping>
        <servlet-name>GetBulkUploadFileServlet</servlet-name>
        <url-pattern>/GetBulkUploadFile</url-pattern>
    </servlet-mapping>

    <session-config>
        <session-timeout>30</session-timeout>
```

```
    </session-config>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

    <taglib>
         <taglib-uri>noaa-form.tld</taglib-uri>
         <taglib-location>noaa-form.tld</taglib-location>
    </taglib>

</web-app>
```

The following tags in the "web.xml" file are noted:

- <web-app></web-app> - contains tags about the Web application that registers and configures servlets and session related parameters
- <servlet></servlet> - contains the tags for the servlet name (<servlet-name></servlet-name>) and the location of the servlet's Java class (<servlet-class></servlet-class>). Servlets may also have initialization parameters (init-param></init-param>), which can be specified with name/value pairs (<param-name></param-name> and <param-value></param-value>).
- <servlet-mapping></servlet-mapping> - contains the tags for the servlet name that was specified above (<servlet-name></servlet-name>) and the URL that will correspond to the execution of the servlet (<url-pattern></url-pattern>). Usually the <servlet> and the <servlet-mapping> tags are required for each Java Servlet.
- <session-config></session-config> - contains tags for configurable session parameters. In the example above, the session timeout for the amount of time to wait before invalidating a session is specified within the <session-config> tag and this example shows 30 minutes is used before a timeout occurs (<session-timeout>30</session-timeout>).
- <welcome-file-list></welcome-file-list> - contains tags for welcome files to use when no URL is specified. In the example above, the JSP index.jsp is called if no URL is specified (e.g., <welcome-file>index.jsp</welcome-file>).
- <taglib></taglib> - contains tags that describe tag libraries used which include the URI (<taglib-uri></taglib-uri>) and the location of the tag library (<taglib-location></taglib-location>) used by the JSPs

8.4 EJB-JAR XML Descriptor

The "ejb-jar.xml" file is the deployment descriptor for Enterprise Java Beans (EJB). During a system build, the IBRDEJB.jar archive is created and then included in the IBRD EAR file. The "ejb-jar.xml" file itself resides in the software development root directory and is stored within the IBRDEJB.jar archive.

This XML file clearly provides information to JRun and the J2EE environment regarding the EJBs that are included in the IBRD application. Every EJB must be explicitly included and if an EJB is ever removed, this file must likewise agree or errors will be generated upon application start up.

Example extracts from the "ejb-jar.xml" file (vertical dots indicate omitted portions):

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE ejb-jar PUBLIC "-//Sun Microsystems, Inc.//DTD Enterprise JavaBeans 1.1//EN"
"http://java.sun.com/j2ee/dtds/ejb-jar_1_1.dtd">
   <ejb-jar id="ejb-jar_ID">
      <description>Generated by Export Tool for Enterprise Java Beans 1.1 version 1.0
from IBM VisualAge for Java version 4.0.</description>
      <display-name>NOAARGDBSystemGroup</display-name>
      <enterprise-beans>
         <session id="BeaconManager">
            <ejb-name>BeaconManager</ejb-name>
            <home>cs.ibrd.complex.ejb.BeaconManagerHome</home>
            <remote>cs.ibrd.complex.ejb.BeaconManager</remote>
            <ejb-class>cs.ibrd.complex.ejb.BeaconManagerBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
         </session>
         <entity id="BeaconRegistration">
            <ejb-name>BeaconRegistration</ejb-name>
            <home>cs.ibrd.complex.ejb.BeaconRegistrationHome</home>
            <remote>cs.ibrd.complex.ejb.BeaconRegistration</remote>
            <ejb-class>cs.ibrd.complex.ejb.BeaconRegistrationBean</ejb-class>
            <persistence-type>Bean</persistence-type>
            <prim-key-class>cs.ibrd.complex.ejb.BeaconRegistrationKey</prim-key-class>
            <reentrant>False</reentrant>
         </entity>
.
.
.
.
.
         <session id="Reporting">
            <ejb-name>Reporting</ejb-name>
            <home>cs.ibrd.reporting.ejb.ReportingHome</home>
            <remote>cs.ibrd.reporting.ejb.Reporting</remote>
            <ejb-class>cs.ibrd.reporting.ejb.ReportingBean</ejb-class>
            <session-type>Stateless</session-type>
            <transaction-type>Container</transaction-type>
         </session>
      </enterprise-beans>
      <assembly-descriptor id="AssemblyDescriptor_ID">
         <container-transaction id="MethodTransaction_1">
            <method id="MethodElement_1">
               <ejb-name>BeaconManager</ejb-name>
               <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
         </container-transaction>
.
.
.
.
.
         <container-transaction id="MethodTransaction_13">
            <method id="MethodElement_13">
               <ejb-name>Reporting</ejb-name>
               <method-name>*</method-name>
            </method>
            <trans-attribute>Required</trans-attribute>
         </container-transaction>
      </assembly-descriptor>
   </ejb-jar>
```


There is no attempt here to expand on the meaning of the tags in the above “ejb-jar.xml” file. With regard to IBRD application development, only unnecessary EJBs associated explicitly with its predecessor (the USA RGDB application), were removed and none were added. Extensive documentation is available elsewhere (i.e., www.java.sun.com) regarding EJBs and if necessary, the reader is left to pursue these details accordingly.

8.5 Access Control List XML Descriptor

The “acl.xml” file provides the specific page mappings for the path info portion (See Section 5.2) of URI requests as well as information pertaining to which user role is allowed access to the page. The “acl.xml” file is found in “\web\WEB-INF” subfolder and during a build it is accordingly stored in the “web-inf” subfolder of the IBRD WAR file.

The JSPs and the servlets within the IBRD main application make use of a specialized component, the dispatcher, which provides flow of control and access control. Dynamic pages in the IBRD system are not accessed directly (static pages may be accessed directly), but are made through a call to the dispatcher. The module that provides the function of the dispatcher is “DispatchServlet.java”. The acl.xml file provides DispatchServlet with the necessary access control information and is loaded into the Servlet Container the first time the DispatchServlet class is called.

The “acl.xml” file contains an access control list for all servlets and specifies the page name which is a parameter passed to the DispatchServlet. In turn, this allows DispatchServlet to determine the JSPs or servlets to run in response to dispatch requests. Each page name definition contains one or more elements which contain a set of roles and the appropriate response to execute. This page name attribute is the parameter passed to the DispatchServlet (e.g., the page name is “UserLogin” in the request syntax http://host/Dispatch?page=UserLogin). Each access tag (or XML element) specifies an allowed role or set of roles for the given page. If the role is an asterisk (“*”), then this access is allowed for all roles. Access is checked one by one in the order represented in the file. The first one that applies is used. If no role access applies, access is denied by default (i.e., the user is presented with the Page Accessed Denied error page).

Example extracts from the “acl.xml” file (vertical dots indicate omitted portions):

```
<acl>
    <page name="UserLogin">
        <access role="*" type="allow" script="/Login.jsp"/>
    </page>

.
.
.
.
.
    <!-- This is the page the user will go to after loggin in -->
    <page name="Start">
        <access role="SHIP SURVEYOR USER" type="allow"
script="/Search?action=display"/>
        <access role="SYSTEM MANAGER" type="allow" script="/Search?action=display"/>
        <access role="SAR USER" type="allow" script="/Search?action=display"/>
        <access role="BLOCK USER" type="allow" script="/BlockHome?action=home"/>
        <access role="BEACON OWNER" type="allow" script="/beaconForm?action=view"/>
    </page>

.
.
.
.
.
    <page name="Update">
        <access role="BEACON OWNER" type="allow" script="/beaconForm?action=update" />
        <access role="BLOCK USER" type="allow" script="/beaconForm=?action=update"/>
        <access role="SYSTEM MANAGER" type="allow"
script="/beaconForm?action=update"/>
    </page>

    <page name="View">
        <access role="BEACON OWNER" type="allow" script="/beaconForm?action=view" />
```

```
        <access role="BLOCK USER" type="allow" script="/beaconForm=?action=view"/>
        <access role="SHIP SURVEYOR USER" type="allow"
script="/beaconForm?action=view"/>
        <access role="SYSTEM MANAGER" type="allow" script="/beaconForm?action=view"/>
        <access role="SAR USER" type="allow" script="/beaconForm=?action=view"/>
    </page>

.
.
.
.
.
    <page name="GetBulkUploadFile">
        <access role="BLOCK USER" type="allow" script="/GetBulkUploadFile.jsp"/>
        <access role="SYSTEM MANAGER" type="allow" script="/GetBulkUploadFile.jsp"/>
    </page>

</acl>
```

The following tags in the "acl.xml" file are noted:

- <page></page> - contains tags and attributes about each JSP or Servlet requested through the dispatcher. The page tag contains a name attribute which specifies the name to match for the dispatch request (e.g., <page name="BeaconForm"></page>) and is the parameter passed to the DispatchServlet. Each page element contains one or more access elements.
- <access /> - contains access information for the roles of the application to allow or not allow the ability to execute the page. Each access element sets access for a role or set of roles. The access tag defines three items for the page: the role that the access rules applies to (e.g., role="SAR USER"), the type of access for this rule (e.g., type="allow"), and the JSP or Servlet with any parameters to call to execute this page (e.g., script="/beaconForm=?action=update").

8.6 Database XML Descriptor

The "dbReg.xml" file contains database field information pertaining to the main registration record table (RegistrationDb406). The "dbReg.xml" file is found in "\web\WEB-INF" subfolder and during a build it is accordingly stored in the "web-inf" subfolder of the IBRD WAR file.

It is used for critical field based operations in a variety of processing situations, but mostly with regard to input and output to the database. The class named BeaconTableData is called to load and reference the actual information. It is important to note that <u>any changes the schema related to field (or column) specifications in the RegistrationDb406 table must also be reflected in this table.</u>

Example extracts from the "dbReg.xml" file (vertical dots indicate omitted portions):

```
<dbReg>

    <declaration name="bcnId15">
        <column name="BcnId15" type="char" size="15" nullable="N" />
    </declaration>
    <declaration name="beaconId">
        <column name="BcnId15" type="char" size="15" nullable="N" />
    </declaration>
    <declaration name="beaconCountryCode">
        <column name="BeaconCountryCode" type="smallint" size="200-800" nullable="N"
/>
    </declaration>
```

```
        <declaration name="cSTACNumber">
            <column name="CSTACNumber" type="varchar" size="10" nullable="Y" />
        </declaration>
        <declaration name="mMSI">
            <column name="MMSI" type="varchar" size="9" nullable="Y" />
        </declaration>
        <declaration name="password">
            <column name="Password" type="varchar" size="16" nullable="N" />
        </declaration>
        <declaration name="beaconRegType">
            <column name="BeaconRegType" type="tinyint" size="0-9" nullable="N" />
        </declaration>
        <declaration name="beaconType">
            <column name="BeaconType" type="varchar" size="32" nullable="N" />
        </declaration>
        <declaration name="beaconActivationMethod">
            <column name="BeaconActivationMethod" type="char" size="4" nullable="Y" />
        </declaration>
.
.
.
.
.
        <declaration name="LastUpdated">
            <column name="LastEditDate" type="datetime" size="8" nullable="N" />
        </declaration>
        <declaration name="LastConfirmationDate">
            <column name="ConfirmPrintDate" type="datetime" size="8" nullable="Y" />
        </declaration>
        <declaration name="BeaconSpecialStatus">
            <column name="SpecialStatus" type="varchar" size="16" nullable="Y" />
        </declaration>
        <declaration name="BeaconRecordStatus">
            <column name="RecordStatus" type="char" size="1" nullable="N" />
        </declaration>
    </dbReg>
```


The following tags in the “dbReg.xml” file are noted:

- <declaration ></declaration> - contains one attribute for the name of the field, which is a name to be used internally by other application modules. It can be noted that multiple declarations and hence internal names may refer to the same field in the database. Each declaration tag also has one column name tag.
- <column name></> - has four attributes pertaining to the field in the database must match the parallel information in the database schema for the RegistrationDb406 table. The attribute “name” is clearly the name of the field, “type” is the SQL data type, “size” is given as bytes (really only used actively for text fields) and “nullable” has the two values “Y” and “N” indicating whether or not the field may be set to NULL.

8.7 Logging XML Descriptor

The “log4j.xml.” file pertains to how logging takes place and where information gets stored. The “log4j.xml” file is found in “\web\WEB-INF” subfolder and during a build it is accordingly stored in the “web-inf” subfolder of the IBRD WAR file.

In fact, very little is currently understood about how the IBRD application specifically performs logging functions. The mechanisms are inherited from the original vendor for the USA RGDB application and the functionality as well the understanding of how it works is limited at best. For the sake of basic information, an example of the XML configuration file is provided below. Perhaps, the more useful information with regard to maintenance in

general can be found in the IBRD System Maintenance Manual where the contents and purpose of the various log files and tables are discussed.

Example “log4j.xml” file (initial deployment for Acceptance Testing):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/"
                     debug="false">

    <appender name="queryAccessLog" class="cs.ibrd.log.QueryAccessLogAppender">
          <layout class="org.apache.log4j.PatternLayout">
                <param      name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"
                />
          </layout>
    </appender>
    <appender name="printEmailFaxLog" class="cs.ibrd.log.PrintEmailFaxLogAppender">
          <layout class="org.apache.log4j.PatternLayout">
                <param      name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"
                />
          </layout>
    </appender>
    <appender name="userAccessLog" class="cs.ibrd.log.UserAccessLogAppender">
          <layout class="org.apache.log4j.PatternLayout">
                <param      name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"
                />
          </layout>
    </appender>
    <appender name="msgLog" class="cs.ibrd.log.MsgLogAppender">
          <layout class="org.apache.log4j.PatternLayout">
                <param      name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"
                />
          </layout>
    </appender>
    <appender name="errorfile" class="org.apache.log4j.DailyRollingFileAppender">
          <!--param name="File" value="./logs/errors.log"/-->
          <param name="File" value="../logs/errors.log"/>
          <param name="DatePattern" value="'.'yyyy-MM-dd-a"/>
          <layout class="org.apache.log4j.PatternLayout">
               <param name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"/>
          </layout>
    </appender>
    <appender name="rollingfile" class="org.apache.log4j.DailyRollingFileAppender">
          <!--param name="File" value="./logs/noaa.log"/-->
          <param name="File" value="../logs/noaa.log"/>
          <param name="DatePattern" value="'.'yyyy-MM-dd-a"/>
          <layout class="org.apache.log4j.PatternLayout">
               <param name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"/>
          </layout>
    </appender>
    <appender name="console" class="org.apache.log4j.ConsoleAppender">
          <layout class="org.apache.log4j.PatternLayout">
               <param name="ConversionPattern"
                            value="=%d{ISO8601}[%t] %-5p %c %x - %m%n"/>
          </layout>
    </appender>
    <category name="UserAccess" additivity="false">
          <priority value="info" />
          <appender-ref ref="userAccessLog"/>
    </category>
    <category name="QueryAccess" additivity="false">
          <priority value="info" />
          <appender-ref ref="queryAccessLog"/>
    </category>
    <category name="PrintEmailFax" additivity="false">
```

```
            <priority value="info" />
            <appender-ref ref="printEmailFaxLog"/>
      </category>
      <category name="cs.ibrd.exception.DAOException" additivity="false">
            <priority value="warn" />
            <appender-ref ref="errorfile"/>
      </category>
      <category name="cs.ibrd.exception" additivity="false">
            <priority value="warn" />
            <appender-ref ref="errorfile"/>
            <appender-ref ref="msgLog"/>
      </category>
         <category name="cs.ibrd.framework" additivity="false">
          <priority value="warn"/>
            <appender-ref ref="errorfile"/>
            <appender-ref ref="msgLog"/>
      </category>
         <category name="cs.ibrd.ui" additivity="false">
          <priority value="warn"/>
            <appender-ref ref="errorfile"/>
            <appender-ref ref="msgLog"/>
      </category>
          <category name="cs.ibrd.taglib.form" additivity="false">
          <priority value="warn"/>
            <appender-ref ref="errorfile"/>
            <appender-ref ref="msgLog"/>
      </category>
      <root>
            <priority value="warn"/>
            <appender-ref ref="console"/>
            <appender-ref ref="rollingfile"/>
            <appender-ref ref="errorfile"/>
            <appender-ref ref="msgLog"/>
      </root>
</log4j:configuration>
```


8.8 Custom Tag Library Descriptors

Custom tags are software elements used in JSP modules to provide functionality in an "HTML tag like" syntax that goes beyond standard HTML capabilities and/or performs application specific processing. The underlying Java code that implements these custom tags can be found in the software development folder "cs\ibrd\taglib\form". The custom tag library descriptors discussed here provide the interface specifications between the usages of the tags in JSP and the underlying code.

There are two text files which are custom tag library descriptors with the names, "noaa-form.tld" and "noaa-framework.tld". Actually, both of these files are XML files, but in these specific cases the extension "tld" is used.

Example "noaa-framework.tld" file (initial deployment for Acceptance Testing):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
<!--
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
//%
//%File Name: noaa-framework.tld
//%
//%Description: This file is a custom tag library definition (tld) that
//% supports many JSP pages.
//%
//%Revisions:
//% 02/05/04 LGL: Added this documentation block.
//% 02/05/04 LGL errors tag: Added optional attribute "language"
//% 08/19/04 LGL errors tag: Added optional attribute "showExtraExpl"
//%
```

```
//%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
-->
<taglib>
    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>NOAAFramework</shortname>
    <tag>
        <name>errors</name>
        <tagclass>cs.ibrd.framework.ErrorList</tagclass>
        <bodycontent>emty</bodycontent>
        <attribute>
            <name>language</name>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
        <attribute>
            <name>showExtraExpl</name>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
    </tag>
</taglib>
```

Example extracts from the “noaa-form.tld” file (vertical dots indicate omitted portions):

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE taglib PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.1//EN"
"http://java.sun.com/j2ee/dtds/web-jsptaglibrary_1_1.dtd">
.
.
.
<taglib>

    <tlibversion>1.0</tlibversion>
    <jspversion>1.1</jspversion>
    <shortname>form</shortname>
    <uri>http://cs.ibrd.ui/tags-form-1.0.0</uri>

    <!-- Text Field Tag -->
    <tag>
        <name>textField</name>
        <tagclass>cs.ibrd.taglib.form.TextFieldTag</tagclass>
        <bodycontent>empty</bodycontent>
        <attribute>
            <name>readOnly</name>
            <required>false</required>
            <rtexprvalue>true</rtexprvalue>
        </attribute>
.
.
.
.
.
        <attribute>
            <name>maxLength</name>
            <required>false</required>
        </attribute>
    </tag>
.
.
.
.
.
    <!--  Radio button list. -->
    <tag>
        <name>radioList</name>
        <tagclass>cs.ibrd.taglib.form.RadioListTag</tagclass>
        <bodycontent>empty</bodycontent>
        <attribute>
            <name>name</name>
            <required>true</required>
        </attribute>
        <attribute>
            <name>mapping</name>
            <required>false</required>
```

```
            </attribute>
            <attribute>
                <name>choiceFont</name>
                <required>true</required>
            </attribute>
            <attribute>
                <name>language</name>
                <rtexprvalue>true</rtexprvalue>
            </attribute>
            <attribute>
                <name>fullList</name>
                <required>false</required>
            </attribute>
        </tag>
.
.
.
.
.
</taglib>
```

The following tags in both custom tag library descriptor files are noted:

- <tag></tag> - contains the specific elements of tags that define the custom tag itself.
- <name></ name > - is the name of the tag (i.e., for JSP usage).
- <tagclass></ tagclass > - contains the name of the Java class, complete with package specification, that supports this custom tag.
- <bodycontent></ bodycontent > - indicates what type of content is to be found in the body of the tag when it is used, if any.
- <attribute></ attribute > - contains a name and various settings for each attribute to associated with this custom tag.

The following two lines of JSP code have been extract from the BeaconStatus.jsp file to provide an example of the usage of a custom tag:

```
<%@ taglib uri="/WEB-INF/noaa-form.tld" prefix="noaaForm" %>
.
.
.
<noaaForm:radioList name="specialStatus" choiceFont="DataFONT" language="<%=language%>" />
```

In the above extract from the “noaaform.tld” tag library descriptor, the specification for the radioList custom tag is given. The following relationships can be observed. In the example the prefix “noaaForm” is defined and linked to the custom tag library descriptor file by the <%@ taglib … %> specification. The usage <noaaForm:radioList … /> provides the name and values for the three “required” attributes. The parallel tag definitions may be easily identified in the <tag> specification. The und“cs.ibrd.taglib.form” folder processes the attribute values accordingly.erlying code, “RadioListTag.java” in the

8.9 Multi-Lingual Functionality

8.9.1 Overview and Basic Mechanisms

As previously mentioned the IBRD application is an adaptation of a very similar software package, the USA RGDB. Perhaps the single most significant difference or enhancement lies in the requirement for the IBRD to provide the user interface in multiple languages. The key to a good design for Internationalization (in Java documentation often abbreviated “i18n”) lies in the

careful separation between what goes on the screen and how it gets there. Ideally, the "how content is displayed" should stay the same while only the content itself dynamically changes with each language. However, the original RGDB application was not designed with Internationalization in mind. On screen information was scattered everywhere appearing at many levels of Java code, HTML files and most extensively within the JSP modules. Since a complete software re-design was not an option, the resulting implementation uses an effective but somewhat "brute force" approach to accomplish the required Internationalization.

Java has some open source classes that offer capabilities and features to support Internationalization, specifically Locale and ResourceBundle. Locale handles a number of settings such as number formats and date/time formats. ResourceBundle uses its "getBundle" method to load content from Properties files (*.properties) for different languages (details are provided in the next section). In the IBRD application the Locale class is left at its default settings which are United States and English. There are perhaps other subtle potential hazards in setting the Locale, but one known problem lies in the fact that the software occasionally parses a date for validity and/or to extract information. Simply put, allowing the Locale to change causes the date format to likewise change and creates problems. As a related point, it is noted that one aspect of Locale typically lies in determining which Properties file the ResourceBundle will load when the "getBundle" method is called. Specially, actively setting the Locale causes "getbundle" to look for file names keyed to each language (e.g., French contains "_fr"). In the IBRD application this mechanism is not used, and the filename to be loaded is provided explicitly with no "assumed keys" expected or provided.

The following bullets summarize the various software elements and components that have been addressed in the Internationalization effort for the IBRD:

- An opening page is provided to select the language for Data Providers (Beacon Owners and National Data Providers). This results in a language option (text string set to "English", "French" etc.) being stored in the Session at the HTTP Request level.
- All pages that originate from the account login page (SAR Users, Ship Surveyors and Database Administrators) are provided in English only. Some pages simply only have English versions and where pages are the same as those used by multi-lingual users (e.g., beacon view/update form), the language for the Session is set to English by default.
- All labeling and general text that appears on the user screen originating from code in JSP files that needs to be multi-lingual reaches the screen by using embedded Java code that passes a "key" (text string) to backend code that in turn returns the required text in the current language for the Session.
- Likewise all multi-lingual labels and general text that appears on the user screen originating from code already in backend Java files is retrieved using a "key" (text string) to obtain the required text in the current language for the Session.

- The translated versions for Image files (predominately buttons) are stored in subfolders under "web\images" with the same name as the text string for the given language as stored in the Session (e.g., "English", "French" etc.). The appropriate image is displayed accordingly by using the Session variable to form the appropriate folder name from which to retrieve the image.
- Java Scripts that provide the set of "available links" at the bottom of every screen are handled similar to image files with language subfolders under the "web\js" folder.
- HTML files in the various languages, predominately "online help files" but others are included, are retrieved from subfolders under the "web\help" folder. These files are discussed further in a subsection just below.
- Letters to be sent via email to end users also are provided in multiple languages and are sent per the current setting for language in the Session. Letters are discussed in more detail below under Section 8.11.

8.9.2 Properties Files for On Screen Text

There are two sets of Properties files that provide the vast majority of the multi-lingual content for the IBRD application. One set of contains error messages (ErrorMessages_xxx.properties) and the other provides for various "on screen" text as well as the labels for many data entry fields (FormLabels_xxx.properties). In each set there is one file for each language supported by the IBRD and the "xxx" in the file name is replaced by the language. For the initial delivery each set consists of four files (English, French, Russian and Spanish).

As mentioned above in the introduction to Section 8, Properties files are text files with individual lines containing a <name/key>=<value> format. The software reads in the key and the associated value which is all the text to the right of the equals sign up to but not including the line feed and carriage return sequence. The data is placed in a Java structure or class called a ResourceBundle which essentially uses "hashmap" or "hashtable" (a set of keys and values) for storage.

It is important to note that for each file in the same set, the keys must all match exactly, or more to the point, all keys coded within and hence expected to be found by the software must be in the file or an error will occur. The order does not matter, but the keys must all be there, with exactly the same spelling, which *is* case sensitive. Clearly, each file will have different values to the right of the equal signs that accordingly supports each language. In order to facilitate tractability of the contents for these files, it is highly recommended that the order of the keys be kept the same for each language file. Example extracts from the various "FormLabel" Properties files follow.

Example extract from "FormLabels_English.properties" (vertical dots indicate omitted portions):

```
# Buttons
Done = Done
Accept = Accept
Login = Login
Cancel = Cancel
.
```

```
.
.
.
.
# General Phases
ClickHere = Click Here
NeedHelp = Need help with this page?
TimeoutReminder = Please note that this page expires within 30 minutes.
.
.
.
.
.
SUCCESS_UPDATE_MSG = The update process has been completed successfully.
WARNING_UPDATE_MSG = The update process has been completed.    However, some of
the information you supplied may be inconsistent with the expected value for that
field.  You should review the information and correct any inconsistencies in the
fields identified with ATTENTION Messages.  Please either resubmit new information
using the 'Update Again' button or click the 'Accept With No Changes' button at
the bottom of the page.
.
.
.
.
.
# Beacon Info DropDowns
121.5_MHz = 121.5 MHz
SART = SART
CAT1 = Category 1 (Automatic or manual)
.
.
.
.
.
ContactUsTextLine2 = Please note that all correspondences can only be serviced
when submitted in English.
```

Example extract from “FormLabels_French.properties” (vertical dots indicate omitted portions):

```
# Buttons
Done = Fini
Accept = Accepter
Login = Connecter
Cancel = Annuler
.
.
.
.
.
# General Phases
ClickHere = Cliquer Ici
NeedHelp = Avoir besoin d&#x0027;aide avec cette page?
TimeoutReminder = Veuillez noter que cette page disparaîtra dans un délai de 30
minutes.
.
.
.
.
.
SUCCESS_UPDATE_MSG = Le processus de mise à jour a été complété avec succès.
WARNING_UPDATE_MSG = Le processus de mise à jour a été complété.  Cependant, une
partie des informations que vous avez fournie peut être en contradiction avec les
valeurs supposées dans certain champs.  Veuillez vérifier vos données et corriger
toutes les valeurs contradictoires dans les champs identifiés par des messages
d'AVERTISSEMENT.  Veuillez soumettre à nouveau les nouvelles données en cliquant
sur le bouton de 'nouvelle mise à jour' ou en cliquant sur le bouton 'accepter
sans changements' en bas de la page.
.
.
.
.
.
```

```
# Beacon Info DropDowns
121.5_MHz = 121.5 MHz
SART = SART
CAT1 = Catégorie 1 (automatique ou manuel)
.
.
.
.
.
ContactUsTextLine2 = Veuillez noter que toutes les correspondances doivent être
soumises en anglais pour être prises en consideration.
```

Example extract from “FormLabels_Spanish.properties” (vertical dots indicate omitted portions):

```
# Buttons
Done = Terminado
Accept = Acepto
Login = Entrar
Cancel = Cancelar
.
.
.
.
.
# General Phases
ClickHere = Haga Click Aquí
NeedHelp = Necesita ayuda con esta página?
TimeoutReminder = Por favor note que su página expirará dentro de 30 minutos.
.
.
.
.
.
SUCCESS_UPDATE_MSG = El proceso de actualizar ha terminado con exito.
WARNING_UPDATE_MSG = El proceso de actualizar está completo.  Pero alguna
información que proporcionó no consiste con lo esperado. Debería revisar la
información y correjir los campos identificados con el mensaje de ATENCION.
Someta nueva información usando el botón 'Actualizar de Nuevo' o haga clic al
botón 'Aceptar Sin Cambios' al inferior de esta página.
.
.
.
.
.
# Beacon Info DropDowns
121.5_MHz = 121.5 MHz
SART = SART
CAT1 = Category 1 (Automatico o manual)
.
.
.
.
.
ContactUsTextLine2 = Por favor note que toda correspondencia puede ser contestada
solo si la somete en Inglés.
```

Example extract from “FormLabels_Russian.properties” (vertical dots indicate omitted portions):

```
# Buttons
Done = &#x0417;&#x0430;&#x0432;&#x0435;&#x0440;&#x0448;&#x0438;&#x0442;&#x044c;
Accept =
&#x0421;&#x043e;&#x0433;&#x043b;&#x0430;&#x0448;&#x0430;&#x044e;&#x0441;&#x044c;
Login = &#x0412;&#x043e;&#x0439;&#x0442;&#x0438;
Cancel = &#x041e;&#x0442;&#x043c;&#x0435;&#x043d;&#x0438;&#x0442;&#x044c;.
.
.
.
.
.
```

```
# General Phases
ClickHere = &#x041d;&#x0430;&#x0436;&#x043c;&#x0438;&#x0442;&#x0435;
&#x0437;&#x0434;&#x0435;&#x0441;&#x044c;
NeedHelp = &#x041d;&#x0443;&#x0436;&#x043d;&#x0430;
&#x043f;&#x043e;&#x043c;&#x043e;&#x0449;&#x044c; c
&#x044d;&#x0442;&#x043e;&#x0439;
&#x0441;&#x0442;&#x0440;&#x0430;&#x043d;&#x0438;&#x0446;&#x044b;?
TimeoutReminder =
&#x041f;&#x043e;&#x0436;&#x0430;&#x043b;&#x0443;&#x0439;&#x0441;&#x0442;&#x0430;
&#x043f;&#x0440;&#x0438;&#x043c;&#x0438;&#x0442;&#x0435; &#x0432;&#x043e;
&#x0432;&#x043d;&#x0438;&#x043c;&#x0430;&#x043d;&#x0438;&#x0435;,
&#x0447;&#x0442;&#x043e; &#x0432;&#x0430;&#x0448;&#x0430;
&#x0441;&#x0435;&#x0441;&#x0441;&#x0438;&#x044f;
&#x0437;&#x0430;&#x043a;&#x043e;&#x043d;&#x0447;&#x0438;&#x0442;&#x0441;&#x044f;
&#x0447;&#x0435;&#x0440;&#x0435;&#x0437; 30
&#x043c;&#x0438;&#x043d;&#x0443;&#x0442;.
.
.
.
.
.
SUCCESS_UPDATE_MSG = Процесс обновления был закончен успешно.
WARNING_UPDATE_MSG = Процесс обновления был закончен с предупреждением (ями). Вы
можете внести исправления и повторно отослать данные с помощью кнопки Обновить. Вы
можете проигнорировать предупреждения и закончить процесс, воспользовавшись
кнопкой Отменить.
.
.
.
.
.
# Beacon Info DropDowns
121.5_MHz = 121.5 MHz
SART = SART
CAT1 = 1 &#x041a;&#x0430;&#x0442;&#x0435;&#x0433;&#x043e;&#x0440;&#x0438;&#x044f;
(&#x0410;&#x0432;&#x0442;&#x043e;&#x043c;&#x0430;&#x0442;&#x0438;&#x0447;&#x0435;&#x
#x0441;&#x043a;&#x0438;&#x0439; &#x0438;&#x043b;&#x0438;
&#x0420;&#x0443;&#x0447;&#x043d;&#x043e;&#x0439;)
.
.
.
.
.
ContactUsTextLine2 =
&#x041f;&#x043e;&#x0436;&#x0430;&#x043b;&#x0443;&#x0439;&#x0441;&#x0442;&#x0430;
&#x043e;&#x0442;&#x043c;&#x0435;&#x0442;&#x044c;&#x0442;&#x0435;:
&#x0412;&#x0441;&#x044f;
&#x043a;&#x043e;&#x0440;&#x0440;&#x0435;&#x0441;&#x043f;&#x043e;&#x043d;&#x0434;&#x0435;&#x
x0435;&#x043d;&#x0446;&#x0438;&#x044f;
&#x0434;&#x043e;&#x043b;&#x0436;&#x043d;&#x0430;
&#x043f;&#x0440;&#x0445;&#x043e;&#x0434;&#x0438;&#x0442;&#x044c;
&#x043d;&#x0430;
&#x0430;&#x043d;&#x0433;&#x043b;&#x0438;&#x0439;&#x0441;&#x043a;&#x043e;&#x043c;
&#x044f;&#x0437;&#x044b;&#x043a;&#x0435;.
```


The above examples highlight several important characteristics that should be observed with regard to these files and somewhat with regard to Properties files in general. Comment lines (ignored when the ResourceBundle is loaded) begin with a pound sign (e.g., "# Buttons"). Keys (to the left of the equal sign) can not contain spaces. Everything to the right of the equals sign is the value, no matter how long. In this document layout, carriage return and line feed sequences appear to occur in the middle of these long value entries, but in the "raw text" originals only one such sequence occurs at the very end.

Furthermore, it can be noted that in the French and Spanish files that special characters or English characters with accents appear. These letters are part of the standard ASCII font sets and as such are displayed here, as well as on web pages

reliably and consistently. However, Russian uses Cyrillic fonts which behave with significant differences. Should an end user be working on a computer fully configured to display and operate in Cyrillic, things may work differently than described the following paragraphs. However, for this development effort there were limited resources as well as the understanding as to how to mimic such computers. More important there is a general expectation that not all users of the Russian interface will have such a computer, in particular in each case with similar settings etc. Put another way, it was determined that a more general approach was needed that would work on any computer running any standard browser with any language providing the native or underlying format.

A brief examination of the "FormLabels_Russian.properties", will quickly suggest that something different is going on here. Most of the multi-lingual text that appears in the IBRD user interface is generated by HTML or JSP modules. The exceptions are the occasional use of browser based pop-up dialog boxes. When the Russian text to be displayed is being generated by HTML or JSP code, using the pure Unicode hexadecimal values is the surest way of getting the desired output. Specifically, Unicode uses four bytes to reference an offset into a given font (ASCII uses only two). The "character images" in the higher references are possible only when a four byte value for an offset is used. This is where Cyrillic and many other non English / European / Latin characters are found. In HTML, and hence likewise in JSP, the format for an absolute character reference takes the form "&#HHHH;" where HHHH is a hexadecimal value (e.g., "043A") and the other characters ("&#" and ";") are leading and trailing delimiters. These sequences are clearly shown in the example above.

More details on the required conversion process from Cyrillic text to HTML Unicode are discussed below in Section 8.9.3. First it is noted that there are some portions in the above example where the actual Cyrillic text is found instead of Unicode. As mentioned above, there are several situations where the software makes use of the browser based pop-up dialog boxes. In these cases the text is handled directly by the browser (and/or operating system settings) and Unicode can not be passed directly. At least when Unicode in the format "\uHHHH" is passed to the browser dialog box, the upper bytes that are the key the to the Unicode capability are apparently ignored. In fact for Cyrillic the raw text does not work either, or at least it does not work in a non-Cyrillic computer environment. It has been assumed that if the computer and/or browser are setup to properly handle Cyrillic fonts that the text will be displayed correctly. *This is still a theory that has yet to be tested*. No perfect solution has yet been found to handle this dichotomy.

There is a similar dichotomy to be noted within the "FormLabels_French.properties" file. A close inspection of this file will reveal that Unicode specifications appear in some places here as well. (See the line "NeedHelp = Avoir besoin d'aide avec cette page?"). There is a different reason for this that falls more into the category of a "special case". Many of the translated labels are "passed" back to JSP code as the text content for the value of an attribute within the HTML tag syntax. For example, the following code sequence appears in many JSP files:

```
<table width=80% summary="table contains help link">
<TR align=right>
<TD>
<A href="<%=helpFileName%>" title="<%=formBean.getLabel(language, "NeedHelp") %>">
<B><%=formBean.getLabel(language, "NeedHelp") %></B></A>
</TD></TR>
</table>
```

In this example value for the "title" attribute is what would cause a problem if the Unicode "&#x0027" is not used. Specifically, in French the actual phrase appears as: "Avoir besoin d'aide avec cette page?" Note that the Unicode sequence "&#x0027" represents a single quote character, which is commonly used for contractions in French phrases. However, since the HTML translates a single quote or a double quote in the same fashion, the single quote if left there in the middle of the phrase, would appear to close the content of the value for this attribute and the rest of the phrase becomes a dangling block of text that the browser (or HTML parser) attempts to handle as the next attribute. A browser error will occur and the desired text will of course not display properly. The solution is to use the Unicode in place of single (or double) quotes for such cases. Inversely, just like our Cyrillic case, this Unicode substitution will not work for elements that are to be displayed in a browser dialog box (e.g., the key "SUCCESS_UPDATE_MSG") and in the cases leaving the single quote intact works best). It should perhaps be noted that this special substitution of Unicode characters is also applied in the French Java Script files (*.js in "web\js\french\").

8.9.3 Conversion of Cyrillic Text to Unicode

As noted above, the majority of Cyrillic text on the IBRD screen is produced by explicit Unicode expressions. Thus, given a set of Cyrillic text represented in the normal readable fonts (i.e., for those who understand Russian) the goal is to convert this text to a valid Unicode sequence used for the IBRD HTML output. It should perhaps be noted that other methods of handling Cyrillic (or other fonts that involve Unicode) certainly exist, and the methods used here simply represent one solution, and not necessarily the best one.

The method used here involves a two step process as given just below. The first step assumes that the Cyrillic text is being provided to the conversion utility in a plain Unicode text file. This means that if the original text is in Microsoft Word or some other editor format, it must be first saved as plain Unicode text (i.e., as used by Microsoft Notepad). Another important assumption is that the Sun Java SDK utilities have been downloaded and installed (See Section 10.2). The two steps involved are:

(1) Use the Java utility named native2ascii.exe found in the SDK subfolder "bin" to convert the Unicode font information into a form of raw Unicode references. For example a command line sequence might appear something like: `native2ascii -encoding Unicode Russian.u.txt Russian.a.txt`

(2) The syntax created in step one looks like "`\u0417\u0430\u0432\u0435\`", but we need it to appear as

"Заве". Each sequence needs to have the slash replaced with ampersand and pound signs, and a semi-colon needs to be added to the end.

The attentive reader is likely to now ask: Why is step (2) needed, as the Java utility is provided by experts in the field and produces something so similar. The answer is: I don't know why, but the second sequence works with the IBRD and the first doesn't. This attentive reader may also note, after thinking about it at least, that while doing a global find and replace will take care of the slashes needing to be changed to ampersands and pound signs, adding a semi-colon is not quite as easy to handle. One way to do it is to include all three characters (i.e., add the semi-colon) in the find and replace operation, but then you need to go back and remove the unnecessary semi-colon at the beginning of each line and add a final one at the end. The solution applied for the IBRD development was a very simple custom program that directly performs Step (2).

8.9.4 HTML Files and Other Support

In the IBRD application, HTML files are used in situations where only static text is required. This is essentially limited to on line help and one or two other elements such as the "Privacy Policy" page. The filenames for "Help" files take the form "xxx_help.htm" where "xxx" is often similar to or the same as the corresponding JSP module.

One important aspect of all HTML files lies in the fact that the translations and in general even the modifications for the "original" English versions have been accomplished using Microsoft Word as the editor (i.e., in HTML mode – files loaded and saved as *.htm). As such, it should be noted that Word often seems to add excess code and comments, as well as occasionally butchering the hyperlinks to images in these files. Clearly, this can make the files a bit cumbersome to work with at the raw HTML level as well as occasionally resulting in some repair to hyperlinks after updates. Regardless, there is a measurable advantage in using Word as the language specific subtleties are dealt with directly by Word, including all the difficulties involved with Cyrillic fonts.

As mentioned above in the summary, HTML files (*.htm), Java Script files (*.js) and image files (*.gif, *.jpg, and *.bmp) are all handled similarly with respect to the use of separate subfolders for each language. When a JSP module needs to provide one of these elements to the end users, the text string for the currently active language is pulled from the HTTP Session hashmap, and used to build a path and filename for the appropriate reference. For example the following embedded Java code appears in the JSP module, "BO_login.jsp" (Beacon Owner Login Page).

```
<%
  // 01/30/04 LGL Support the language choice
  String language = (String)
request.getSession().getAttribute(RGDBConstants.SESSION_ACTIVE_LANGUAGE);
  String helpFileName = "./help/" + language + "/BO_login_help.htm";
  String loginImageName = "./images/" + language + "/login.gif";
  String cancelImageName = "./images/" + language + "/cancel.gif";
  String footerFileName = "./js/" + language + "/welcome_footer.js";
```

```
%>
```

Clearly, a local variable named “language” is assigned and used to form filenames (which in clued paths) accordingly. The following code examples from further down in the same JSP module demonstrate the simple usage for these references.

```
<A href="<%=helpFileName%>" title="<%=formBean.getLabel(language, "NeedHelp")
%>"><B><%=formBean.getLabel(language, "NeedHelp") %></B></A>
.
.
.
.
.
<a href= "javascript:document.forms[0].submit();"><img border="0"
src="<%=loginImageName%>" alt="<%=formBean.getLabel(language, "Login") %>
<%=formBean.getLabel(language, "Button") %>" width=90 height=20
TABINDEX=3></a>   
<A href="Dispatch?page=Index"><IMG border="0" src="<%=cancelImageName%>"
alt="<%=formBean.getLabel(language, "Cancel") %> <%=formBean.getLabel(language,
"Button") %>" width=90 height=20 TABINDEX=4></A>
.
.
.
.
.
<SCRIPT LANGUAGE="JavaScript" src="<%=footerFileName%>">
```

8.10 Document Manager Properties File

The DocumentManager.properties file, resides singularly in the IBRD JRun Server folder (e.g., “C:\JRun4\servers\IBRD”), and maintains information on email, in particular the templates for formatting letters sent to users. This is the only XML and/or Properties file that is not “compiled” into the archive file (IBRD.EAR) that comprises the deployable IBRD application software.

Example “DocumentManager.properties" file (initial deployment for Acceptance Testing):

```
################################################################################
################################################################################
##
## NOTE: ANT scripts replace the KEYWORDs such as the root directory. Do not
##       change the KEYWORD name.
##
################################################################################
################################################################################

PrintDebug.printMessagesFlag=true
PrintDebug.logFileName=c:/jrun4/servers/IBRD/dispatch/temp/printDebugLog.txt
RuntimeExecProcessor.consumeBuffers=false

################################################################################
#
# DocumentManager Properties
#
################################################################################

# This boolean flag determines if the temporary files should
# be removed after they are dispatched. A setting of "false"
# leaves the file in the below directory
DocumentManager.removeDocumentsAfterUse=false

# This directory hosts the temporary files.
# IMPORTANT: THIS SHOULD HAVE ALL DOUBLE REVERSE SLASHES
# Example: c:\\jrun4\\servers\\default\\IBRD-ear\\dispatch\\temp
DocumentManager.documentDirectory=c:\\jrun4\\servers\\IBRD\\dispatch\\temp

# DO NOT change this property value
DocumentManager.simulatedNormalMode=false

################################################################################
#
# EmailDispatcher Properties
#
################################################################################

# A common name to easily identify the email server (used only for logging)
EmailDispatcher.commonName=IBRDEmailServer1

# SMTP Host server name (network id)
EmailDispatcher.smtpHost=nes3.nesdis.noaa.gov

# Email address to be used as "from" address for all the emails
EmailDispatcher.fromAddress=IBRD@noaa.gov

# Flag to use the real email address on Beacon Account or the testaddress (see below)
# as "to" address. testaddress is effective only when the flag is set to false
EmailDispatcher.sendToRealUser=true
# EmailDispatcher.testAddress=IBRD@noaa.gov


################################################################################
#
# DocumentAuthor Properties
#
#     Properties point to other files containing content for letters
#
################################################################################

# Request for Confirmation Letter templates
DocumentAuthor.ConfReqEmailFileEnglish=c:/jrun4/servers/IBRD/dispatch/templates/confir
mation_request_email_English.txt
DocumentAuthor.ConfReqEmailFileFrench=c:/jrun4/servers/IBRD/dispatch/templates/confirm
ation_request_email_French.txt
DocumentAuthor.ConfReqEmailFileRussian=c:/jrun4/servers/IBRD/dispatch/templates/confir
mation_request_email_Russian.txt
DocumentAuthor.ConfReqEmailFileSpanish=c:/jrun4/servers/IBRD/dispatch/templates/confir
mation_request_email_Spanish.txt
# Registration "Information Provided" Letter templates
```

```
DocumentAuthor.RegEmailFileEnglish=c:/jrun4/servers/IBRD/dispatch/templates/registrati
on_email_English.txt
DocumentAuthor.RegEmailFileFrench=c:/jrun4/servers/IBRD/dispatch/templates/registratio
n_email_French.txt
DocumentAuthor.RegEmailFileRussian=c:/jrun4/servers/IBRD/dispatch/templates/registrati
on_email_Russian.txt
DocumentAuthor.RegEmailFileSpanish=c:/jrun4/servers/IBRD/dispatch/templates/registrati
on_email_Spanish.txt
```

8.11 Letters and Templates

There are two letters automatically generated by the IBRD application, one sent after a record is added or modified and the other to request users to confirm data that has not been updated in two years. Similar discussion and examples are provided in the IBRD System Maintenance Manual with the emphasis here on the underlying software aspects.

The email "text body" files are simple text files that contain the content of the letters to be sent to end users. Only users who provide a valid email address can be sent messages. There are actually four files for each letter format, one for each of the languages supported by the IBRD: English, French, Russian and Spanish. The files names and locations are stored in the DocumentManager.properties file. It should be noted that the salutation line (i.e.., Dear <beacon owner>), as well as the URL (found in SystemCfg) are added by the software, at the beginning and ending respectively, when the email is generated. Finally, each email includes a segment in the body of the email consisting of a simple field by field text listing of the registration information currently on record in the database.

This all takes place in the Java class modules named "DocumentAuthor.java", "DocumentManagerBean.java" and "DocumentAuthorTextOnly.java". Specifically, when an email needs to be generated, "ProcessRequest" under the dispatcher within "DocumentManagerBean" build and sends the necessary email. Based on the settings in "DocumentManager.properties", the individual various pieces are pulled from within "DocumentAuthor", with the specific contents being in part managed by "DocumentAuthorTextOnly".

Once a given email has been constructed by these modules, "DocumentManagerBean" passes it to the resident or local email Server (identified in "DocumentManager.properties") via a call to the method named "sendMail" from the "EmailDispatcher" Class. Ultimately, the call to "Transport.send(msg)" in "sendMail" invokes the native Java capabilities from the "javax.mail" Class to send the email.

8.12 Request for Confirmation Process

This software component determines the Beacons for which a Request for Confirmation Letter is needed and transmits them accordingly to Beacon Owners. It is implemented as a Java program and runs as a background process that is invoked by the Windows 2000 operating system task scheduler (usually every 24 hours).

The underlying goal is to determine whether or not a request for confirmation needs to be generated in association with a given registration record. Records in the applicable "date range" as are determined as follows:

1. Add two years to the ConfirmationPrintDate and include records when the difference between today and this computed date is less than or equal to the number of days in the SystemCfg table (field "CONFIRMATION REQUEST TIME"). It is useful to note that "really old" dates will result in negative values and hence be included.
2. Given that the above "date range" criteria is met for a given record the following configuration and status conditions must also be met:
    a. Corresponding ComfirmationRequired from MidInfoCfg (record where mid matches the beacon country code) is "Yes".
    b. ConfirmationStatus field is not NULL, "SENT" or "UDEL"
    c. SpecialStatus is NULL (i.e., Normal status)
    d. Record must have an "valid" email address field (not NULL or empty)

The actual query used within the software at time of installation reads as follows:

```
select bcnId15, OwnerName, a.BeaconRegType, BeaconRegName, EmailAddress
from RegistrationDB406 a, MidInfoCfg b, BeaconRegTypeCfg c
where a.BeaconRegType = c.BeaconRegType
and a.BeaconCountryCode = b.mid
and b.ConfirmationRequired = 'Y'
and (ConfirmationStatus IS NULL or ConfirmationStatus not in ('SENT', 'UDEL'))
and a.SpecialStatus IS NULL
and datediff(DAY, GETDATE(), dateadd(YEAR, 2, ConfirmPrintDate)) <=
(select SystemCfgValue from SystemCfg where SystemCfgName = 'CONFIRMATION REQUEST TIME')
order by bcnid15
```

If desired, the above query could be manually run at anytime from "SQL Query Analyzer" to determine which records should be picked up in the next run. It is useful to note that the IBRD implementation does not include records where ConfirmationStatus is NULL, as was the case with the original RGDB code upon which the RGDB is based. In fact all new records are assigned a default status of "CHGE" and hence a ConfirmationStatus of NULL should not really occur.

Furthermore, no matter when an update is made to a record (other than by this "request generation" process itself) the IBRD always sets the ConfirmationStatus to "CHGE" as well as setting the ConfirmPrintDate to the current date and time. The status of "CFRM" is reserved exclusively for when a user explicitly performs an online acknowledgement. As such, the whole business of generating "requests for confirmation" really hinges almost exclusively on the ConfirmPrintDate.

Finally, it is perhaps useful to note that there is no attempt here to maintain a two-year renewal cycle that expires on any given day of the year like other types of registrations (e.g., car, vessel etc.) are often implemented. A "request for confirmation" is only generated when a user has not made any update for a period of two years (minus the specified period in SystemCfg to be precise) and any update at all will start the two year cycle over beginning at the new date.

8.13 FileArcPurge Process

A special purpose application is used to purge files created by the IBRD software. This application, named IBRDFileArcPurge.exe, runs as a scheduled task and uses a configuration

table in the IBRD database which indicates the folders to be checked and the "age out" criteria of files to be purged.

The program is written in Visual Basic (Version 6.0), and is really a very simple set of code that employs Microsoft supplied mechanisms (class named FileSystemObject) to perform the necessary file management actions. The folders (or file paths) and the number of days to retain a given type of file are all configured using the table named "DbmnFileArcPurgeCfg". This table and some example settings are provided in the IBRD System Maintenance Manual.

8.14 ArcPurgeTables Process

The table archive application (IBRDArcPurgeTables.exe) takes care of archiving data from various tables in the IBRD database. This program moved "aged out" records to similar tables found in a second database which is the IBRDArchive database. The number of days that records are kept in each table is configured by setting appropriate values in the SystemCfg table. The SystemCfg table and various example settings are provided in the IBRD System Maintenance Manual.

The central program is written in Visual Basic (Version 6.0). The code here simply gets values from SystemCfg and calls a set of SQL Stored Procedures which actually do the bulk of the work. As the following table indicates, there are eight tables identified for archival and hence eight associated Stored Procedures. Clearly there are and eight corresponding entries in the SystemCfg table and in the table below default values are given that will be applied should no entry be found in SystemCfg.

Table Name	Stored Procedure Name	"Days To Keep" Default Value
OperMsgLog	ArcPurgeIBRDOperMsgLog	180
LogUserAccess	ArcPurgeIBRDLogUserAccess	180
LogQueryAccess	ArcPurgeIBRDLogQueryAccess	180
LogPrtEmailFax	ArcPurgeIBRDLogPrtEmailFax	180
SarTransactionLog	ArcPurgeIBRDSARTransactionLog	180
ConfigChangeLog	ArcPurgeIBRDConfigChangeLog	90
Feedback	ArcPurgeIBRDFeedback	365
RegistrationDb406	ArcPurgeIBRDRegistrationDb406	3650

An example Stored Procedure is listed as follows. Clearly the logic is fairly simple and each one is quite similar. Nonetheless, a different one is needed for each table as different field names are accordingly referenced, perhaps most noteworthy the specific date/time field used for "age out" comparisons.

```
CREATE PROCEDURE ArcPurgeIBRDConfigChangeLog
@daysago int=180
AS
declare @go datetime,@max int,@i int,@id int,@t datetime
set deadlock_priority low
set nocount on
begin tran
  set identity_insert IBRDArchive.dbo.ConfigChangeLog on
  select @go=getdate(),@max=40,@i=0,@id=null,@t=min(ChangeTime)from IBRD.dbo.ConfigChangeLog
  select @id=min(ChangeId)from IBRD.dbo.ConfigChangeLog where ChangeTime=@t
  while @id is not null and @t is not null and @t<dateadd(day,-@daysago,@go) and
getdate()<dateadd(second,@max,@go)
    begin
```

```
      insert IBRDArchive.dbo.ConfigChangeLog(ChangeId, ChangeIdLnk, OprMsgId, ChangeTime,
DBLevel, ChangeType,
            TempFlag, TableName, RowId, FieldName, OldValue, NewValue, Program, UserId,
SubsysId, TrackingId,
            Remarks, SessionId, SessionDate) select * from IBRD.dbo.ConfigChangeLog where
ChangeId=@id
      if @@error=0 delete IBRD.dbo.ConfigChangeLog where ChangeId=@id
      else delete IBRDArchive.dbo.ConfigChangeLog where ChangeId=@id
      select @i=@i+1,@id=null,@t=min(ChangeTime)from IBRD.dbo.ConfigChangeLog
      select @id=min(ChangeId)from IBRD.dbo.ConfigChangeLog where ChangeTime=@t
    end
  if @i>0 select ltrim(str(@i))+' ConfigChangeLog '+ltrim(str(datediff(ms,@go,getdate())))+'
ms'
  set identity_insert IBRDArchive.dbo.ConfigChangeLog off
commit tran

set deadlock_priority normal
set nocount off
GO
```

The Stored Procedure essential loops through the available records in the original table, identifies the next candidate to be archived, moves this record to the corresponding table in the archive and if no error occurs, deletes it from the original.

8.15 Beacon Decode Process

The IBRD System has requires a mechanism to perform the validation and decoding of the 15 character hexadecimal identification code assigned to a 406 MHz Emergency Beacon (or Beacon ID). An existing software package written in C++ (used at the USMCC) is employed here to supply this capability. This package adheres to the Specification for Cospas-Sarsat 406 MHz Distress Beacons (C/S T.001). The interface between this existing package and the IBRD application is described in this document but the details regarding the implementation of the underlying package are not.

The external Validation/Decode component is provided to the IBRD as a dynamic link library (DLL) with the Beacon ID being passed in an input and the desired information extracted and returned accordingly. The IBRD System that calls this component is coded in Java and the Java Native Interface (JNI) is used to interface with this C++ module. Specifically, the component is within a "wrapper" DLL with the actual code compiled as a C++ library.

The following tables provide in actual input and output parameters passed to and from the Validation/Decode component.

Input:

Name	C++ Declaration	Description
BeaconId	char str[15]	15 character hexadecimal identification code.

Values returned:

Name	C++ Declaration	Description
Valid	int	Indicates if passed Beacon Id is valid (1) or invalid (0).
InvalidMessage	char str[80]	Reason that Beacon Id fails validation.
BeaconRegType	int	Value for the BeaconRegType column in the IBRD RegistrationDb406 table that corresponds to the Registration Type of the Beacon. This must contain a Registration

Name	C++ Declaration	Description
		Type.
BeaconType	char str[34]	Value for the BeaconType column in the IBRD RegistrationDb406 table that corresponds to the Type of the Beacon. This must contain a valid Beacon type.
BeaconActivationMethod	char str[6]	Value for the BeaconActivationMethod column in the IBRD RegistrationDb406 table that corresponds to the Category of the Beacon. This value will be an empty string (“”) if the Category can not be determined.
BeaconManufacturer	char str[50]	Value for BeaconManufacturer column in the IBRD RegistrationDb406 table that corresponds to the Manufacturer of the Beacon. This value will be an empty string (“”) if the Manufacturer cannot be determined.[1]
BeaconModel	char str[34]	Value for the BeaconModel column in the IBRD RegistrationDb406 table that corresponds to the Manufacturer’s Model Number for the Beacon. This value will be an empty string (“”) if the Model Number cannot be determined. [1]
BeaconCountryCode	int	Value for the BeaconCountryCode column in the IBRD RegistrationDb406 table that corresponds to the Country Code of the Beacon. This must contain a valid Country Code.
TypeApprovalNumber	int	Cospas-Sarsat beacon type approval number. This value will be a zero (“0”) if the Number cannot be determined

1. Manufacturer and Model Number are generally not available from a decode operation within the IBRD as this is based on National coding schemes, and as such the USMCC based module only provides these fields for USA coded beacons (which are not stored in the IBRD).

The supporting code is found in the Class named “DecodedBeacon.java” (under the *cs\ibrd\complex* folder) and is called as needed throughout the Java code of the IBRD application. The DLL is named IBRD_BeaconDecode.dll and must be stored in the JRun4 Servers library folder (e.g., “to C:\JRun4\servers\lib”).

- END OF SECTION 8 -

9. SOFTWARE MODIFICATIONS

The following sections detail the requirements and methodology for making software modifications to the IBRD application. In general, the requirements for the development environment are very similar to those of the run time one. In particular, a Java Virtual Machine (i.e., JRun4) and SQL Server are needed, along with enough "horsepower" (disk space and memory) to support the basics as well as the debugging environment. COTS components required include:

- JRun4 (Service Pack 1 or higher)
- SQL Server 2000 (Service Pack 3 or higher)
- Netbeans IDE (Version 3.4 or higher)
- Java SDK (Version 1.4.0.02 or higher)
- Jakarta Ant

9.1 IDE Installation

By definition an IDE provides the means to view and modify source code as well as build new application packages. Perhaps the most critical facility an IDE can provide is the mechanism for setting breakpoints, stepping through and otherwise "debugging" the code. Although a number of choices are available for this type of activity, it is likely that using the same IDE that was used for IBRD development will result in the most effective or efficient selection. A free version of the Netbeans package has been used, and in particular Netbeans IDE Version 3.4 (later versions are available). Netbeans can be downloaded at www.netbeans.org.

If Netbeans is selected as the IDE, one more element is needed to allow for full use of the facilities under which the IBRD was developed. Specifically you need to install a package called "Jakarta Ant". This can be downloaded off the Internet, but as the appropriate location of the source web site could not be readily located at the time this document was compiled. This package provides for the use of XML based command scripts which invoke various utilities used to deploy and debug this source code. In effect, these script mechanisms are analogous in many ways to a "make" file used in other IDE packages like Microsoft's Visual Studio.

9.2 Using Netbeans

There is definitely no intention to provide a complete tutorial here on using Netbeans. However, a few basics to "get started" are provided, leaving the actual learning curve to the reader.

When you run Netbeans for the first time, you will need to "mount the filesystem" where you have stored the IBRD source code. You start this process by using the associated selection under the "File" menu (File->Mount FileSystem...). From the dialog box that appears select "Local Directory", click the "Next" button and then proceed to navigate to the subfolder where you placed all the source code files (done just above in Part II Section 1). When you

have properly highlighted a folder name, the “Finish” button will become “available” to complete the process. Once the directory is “mounted” you can open subfolders by clicking on the circles to the left of each folder name. As you navigate down, you eventually find files (as opposed to subfolders) which can be opened by double clicks, or by using a right click and selecting the appropriate operation from a list of operations.

With regard to using Netbeans to look at files and modify them, no further comment is made here, allowing for using “help” or advice directly from the vendor via the Internet. The software is reasonably intuitive. Likewise, with regard to Java code and the full J2EE programming environment of which the IBRD uses nearly every basic feature, from Java code, to Java scripts to custom tag libraries, no further comments are made. The one area that it is pertinent to expand on at least to some degree involves the processes of debugging and recompiling since the support for these functions are perhaps somewhat unique to the setup for this application.

In order to accomplish tasks such as debugging and recompiling you need to use the Jakarta Ant commands. The Jakarta Ant commands themselves appear under a folder named “build” that is not really a folder at all. There are actually several folders with names like “build” which does confuse the issue even more, but you simply need to select the one that has the extra symbol in the icon and reads “Anonymous Ant Project” when the cursor is used to get the “tool tip” text. Click on the circle to “open” the “folder”. The “opened folder” lists the possible commands, rather than files, in this case. It is important to note that you should not use the similar looking commands from the Netbeans “Build” menu. It is perhaps useful to mention here that the filed named “build.xml” actually contains the definitions for the commands that are listed. As such the command actions can be edited, new ones added or whatever.

Before discussing the individual commands, a bit of background needs to be covered. The application software used in runtime installation by copying files into the JRun servers subfolder (e.g., “C:\JRun4\Servers\IBRD\”), is in the “release” format, as opposed to “debug” format. The release package actually consists of two files, “IBRD.ear” and “IBRD_util.jar. These two files are actually compressed archives of many other files (much like a WinZip file). These files can contain many files including other Jar files, War files (another type of archive) and individual class files, which are the “compiled” or “object” files for Java code. The “debug” format is not compressed and exists in the same server subfolder (e.g., “C:\JRun4\Servers\IBRD\”) as a subfolder named “IBRD-ear”. It is very important to note that if both are there at the same time, that JRun will attempt to load everything and serious errors will result. In effect, JRun will try to load any possible runtime files (archives, class files, scripts etc.) from every subfolder, no matter how deep in the tree, under the “...Servers\IBRD\” subfolder. Folders that contain no “runtime” files, simply log minor errors, but otherwise everything available gets loaded.

Different Jakarta Ant commands within the “build” XML file will generate “release” and “debug” formats accordingly. The debug format will be “deployed” directly to the runtime directory (e.g., “C:\JRun4\Servers\IBRD\”). If you did not install JRun at “C\JRun4\”, the text file named “build_config.properties” will need to change accordingly for some of the commands to work properly. Release formats get created in a subdirectory right there with the source code named “build” and must then be manually copied to the server subfolder.

Now, returning to the “build” folder in Netbeans that has the extra symbol, a click on the circle to the left to see the available commands. This does not really “open” a “folder” but rather it opens an XML file containing the “Ant commands”. By using a right click and selecting “execute” the various commands can be invoked. The commands used most, if not almost exclusively, are:
ear - compiles code in “release” format and puts it into the local “build” subfolder
clean - removes the entire local “build” subfolder
cleanDeploy - removes the “debug” IBRD-ear subfolder from the server subfolder
deploy - compiles code and puts it into the “debug” format IBRD-ear subfolder
startDebug - starts the IBRD JRun Server in debug mode
stop - stops the IBRD JRun Server (regardless of start-up mode)
start - starts the IBRD JRun Server in standard (non-debug) mode

It is highly recommended that “clean” be used most of the time before building new code. This is simply a matter of experience with problems that come from modules (or classes) not always properly compiling in sync with each other. This is because all code interdependencies are not generally respected when recompiling subsets of files, and the only way to be sure that you have all the code in full sync is to do a full compile, which in this case requires a “clean” operation as the first step. The “clean” command is required whether you are updating a “release” format or a “debug” format package. This might seem confusing, as “cleanDeploy” would appear to take care of “debug” packages. Simply put, it does not. The class files (compiled Java code) are always created in the local “build” subfolder and the “deploy” operation simply takes the next step after compiling, of placing the results (uncompressed) into the IBRD-ear subfolder.

Finally, the commands associated with running the application from within Netbeans should be self-explanatory but some critical points need to be made. The code can of course always be run by invoking the proper commands from the “JRun Launcher”. The two commands “start” and “stop” are essentially the same as those used in the “JRun Launcher” and are really provided as a convenience. The “startDebug” command is significantly more specialized. It starts the IBRD JRun servers in a mode where the Netbeans IDE can be “attached to the process” allowing for the critical debugging capabilities of setting breakpoints and stepping through source code. In order to use this special mode, Netbeans must be told to “attach” to the process. Once the IBRD JRun server has been started using the “startDebug” command, use the Netbeans menu to invoke Debug->Start Session->Attach. A dialog window should appear with various text box settings. The first boxes are set by default and the last box titled “Name” should contain the number “5000" which is the special port to which Netbeans should “attach”. If all goes well, setting a breakpoint and stepping through code should work.

A couple more pointers might be made with regard to “working in debug mode”. Sometimes, ending a debug session does not work smoothly. Ideally, the Ant command “stop” takes care of everything but for cleaner endings it is better to first make sure the code is not paused but is “running”, then use Debug->Finish to end the “attached Session”, and then last of all use the Ant command “stop” to halt the IBRD JRun server.

- END OF SECTION 9 -

page left blank

10. IBRD SYSTEM INSTALLATION

10.1 Installing the Database

The essential operation consists of simply "attaching" database files to the resident SQL Server 2000. Although the SQL Server coexists on the same computer as the application software in most of the test and/or operational environments at NOAA, this is not required. What is required is that the SQL Server be available such that it is possible to configure a valid JDBC connection between the application and the new SQL Server database. (JDBC stands for Java Database Connectivity which is analogous to ODBC (Open Database Connectivity) connections often used for other types of applications).

Step 1.1 – Copy database files

Obtain the four database files from the distribution subfolder named "Database" and copy them to an appropriate folder for SQL Databases. File names:

IBRD_Data.MDF
IBRD_Log.MDF
IBRDArchive_Data.MDF
IBRDArchive_Log.MDF

This can be on the local computer, an independent disk array, a network accessible drive etc . (A typical default location used by SQL Server might look like C:\Program Files\Microsoft SQL Server\MSSQL\Data).

NOTE: If the files are from a CD, the "read-only" attributes need to be changed.

Step 1.2 – Attach the IBRD database

First you need to run the application "SQL Query Analyzer" and connect to the SQL Server that will support this database. Leave the query/result window that pops up connected to the "Master" database. Enter a command similar to the following, replacing the path information given here with the actual location where the files were stored in Step 1.1 above.

```
sp_attach_db'IBRD',
          'C:\SQLData\IBRD_Data.MDF',
          'C:\SQLData\IBRD_Log.LDF'

sp_attach_db'IBRDArchive',
          'C:\SQLData\IBRDArchive_Data.MDF',
          'C:\SQLData\IBRDArchive_Log.LDF'
```

Step 1.3 – Create the "IBRD" SQL Login

If this is a new SQL Server (just installed) you will likely need to change the default settings with regard to allowable methods for "authentication". The key is that the database must be configured for "mixed authentication".

Next you may need to remove the existing IBRD User from the database you just attached. In effect, the next step creates this user for the new environment and the one already associated with the database that came from the development environment must be removed.

Now you need to run the application "SQL Enterprise Manager" and navigate from the "Console Root" down to the "Security" settings for the SQL Server that will support the new IBRD database. Using a right click on "Logins", add a "New Login" named "IBRD". The default database for this user should be set to IBRD. This login should be configured to use "SQL Server Authentication" with the password being "ibrd" (note: password is all lower case and login name is all upper case). Under the Database Access Tab check the "Permit" box for the new IBRD and IBRDArchive databases, and in the associated window below for each (for "Permit in Database Role") check the boxes for both "public" and "db_owner". For all the other user/login settings, the defaults should suffice unless otherwise dictated by other factors in the destination environment.

10.2 Installing the Application and Related COTS

Step 2.1 – Install JRun4 and Supporting Java Utilities

The JRun4 application must be installed (along with the latest service pack(s)) on the computer designated as the App Server for this installation. Since this is a third party package, guidance regarding installation is provided elsewhere, but it is noted here that you *must* install a JRE (Java Runtime Environment) package *before* you can install JRun. The JRE does not need to be purchased, but rather it can and should be downloaded from the Sun Java web site pages (go to http://java.sun.com). Specifically, you should download and install the Java SDK package from the Sun web site as this contains a JRE as well as additional elements (on main page under "Popular Downloads" select "JRSE 1.4.2 SDK" or similar). The complete path where the SDK has been installed will be needed further below so make note of it accordingly. It is recommended that you install the JRun4 package itself on the "C:" drive in a folder named simply "JRun4", full path "C:\JRun4" (i.e., to accomplish this, override the default path involving "C:\Program Files\..." and enter your own destination folder during the first part of the JRun4 install sequence). This is not explicitly required, but it will simplify the basic steps for the run time installation here, and more important it will eliminate a difficult detail or two if there is an intention to work with the source code using the same Netbeans environment under which this code has been developed.

2.1.1 - Install Java SDK (recommended path: C:\Java\...)
2.1.2 - Install JRun4 (*highly* recommended path: C:\JRun4)

NOTE: The login and password for the JRun4 Management Console are created when JRun4 is installed. This utility will need to be run to perform the below steps related

to configuring the IBRD JRun Server and as such the login and *password for the JRun4 Management Console must be carefully recorded*.

Step 2.2 – Install Application and Supporting Files

This step requires several simple, but very specific, file copy operations. All files required for this step are located in distribution subfolder named "Runtime". For simplicity it is assumed here that JRun4 was installed with the path "C:\JRun4" and that the Java SDK was installed in "C:\Java\". The notation "<SDK path>" indicates where you need to insert the actual SDK path (e.g., "j2sdk-1_4_0_02").

2.2.1 - Copy entire IBRD folder (and subfolders) to C:\JRun4\Servers

2.2.2 - Copy IBRD_BeaconDecode.dll to C:\Java\<SDK path>\jre\bin

2.2.3 - Copy log4j-1.2.4.jar to C:\JRun4\servers\lib

2.2.4 - Copy xerces.jar to C:\JRun4\servers\lib

NOTE: If the files are from a CD, the "read-only" attributes need to be changed.

Step 2.3 – Configuration of the IBRD JRun Server

This step essentially makes it possible to "run" this Java based web site. Once the configuration is established properly, the IBRD web site can be accessed and used on the local machine. The web site can be made available to the actual Internet by making proper additional settings in JRun, or by installing the Apache HTTP Server package as discussed in Step 2.4. (The only methodology for connecting the site to the Internet discussed further here uses Apache as the "Web" Server software layer).

In order to proceed with the configuration of the IBRD JRun Server, we need to access the JRun Management Console (several steps are actually required to do this), and then use this utility to make settings that will support the new IBRD JRun Server that we are about to create. Throughout this process (as well as for using the IBRD application) you will need to have a Java Script enabled browser. It might also be noted that the IBRD itself will definitely work best with Internet Explorer (Version 5.0 or higher).

2.3.1 - Run the JRun Launcher

Start->Programs->Macromedia JRun 4->JRun Launcher

2.3.2 - Run the Admin JRun Server

In the dialog box that appears (one of the two windows to appear), highlight the JRun Server named "admin", and then click the "Start" button. (If you just installed JRun4, the Admin Server may already be running).

2.3.3 - Run the JRun Management Console

Now that the "admin" Server is running, you can start the JRun Management Console with: Start->Programs-> Macromedia JRun 4->JRun Management Console. (If you just installed JRun4, the JRun Management Console may already be running).

2.3.4 - Login

NOTE: The login and password was created for this interface when JRun4 was installed.

2.3.5 - Create New Server

This is the first link on top bar of the screen. Accept the default setting for Host Name (i.e., localhost) and type in "IBRD" for the JRun Server

Name. By simply clicking in the empty box below, the JRun Server Folder should then automatically default to "{jrun.home}/servers/IBRD" which corresponds to the folder we created in Step 2.1.1 above. When ready, click on the "Create Server" button. When the next window pops up, leave the port numbers etc. as they are, and select Finish to complete the process.

2.3.6 - Start IBRD Server

You should now see your new IBRD Server on the list of available servers (on the "Home" screen of the JRun Management Console). In order to configure it further, we must start the new server. Click the arrow/triangle symbol to the left of the name (or double click the IBRD Server name and then select the last link listed) to "Start the Server".

2.3.7 - Configure J2EE Components

If not already there, go to the IBRD Server to further configure the related settings. The only thing you need to accomplish here is the removal of the "default-ear" component. To do this, you must first delete the corresponding subfolder from the C:\JRun4\Servers\IBRD folder. (This subfolder was generated by JRun4 when we created the server in Step 2.3.5 above). Once you have deleted the subfolder, you can click the X button next to the "default-ear" component to delete it. (You may get a minor "error" message, but when you select the option to return to the J2EE components listing, the "default-ear" component should be deleted accordingly). The IBRD and FlashRemotingEar should be left in place on the list.

2.3.8 - Configure a Resource for the Database Connection

The critical action here is to create the JDBC connection to the SQL Server database. Start by clicking the "Resources" link at the top. In the displayed dialog box for "adding" a data source, type IBRD in for the Data Source Name, leave the Database Driver defaulted to "Not-Listed" and click the "add" button. In the dialog box that pops up next for Data Source Settings make the following settings (read all notes below also, ***before*** clicking "Submit"):

Data Source Name: `IBRD`
JNDI Name: `jdbc/IBRD`
Driver Class Name:
`macromedia.jdbc.MacromediaDriver`
URL:
`jdbc:macromedia:sqlserver://<computer_name>:1433;databaseName=IBRD;SelectMethod=cursor;`
Description: `IBRD Data Source`
Pool Connections: `"checked"`
Pool Statements: `"unchecked"`
Native Results: `"checked"`
User Name: `IBRD`
Password: `ibrd`
Verify Password: `ibrd`

Note that the User Name is upper case and that the password entries are lower case, just like when we added this Login/User in Step 1.3 of the database installation above. Also note the URL entry must be typed in exactly as shown with no spaces and that the <computer_name> portion of the given string must be properly filled in with the correct information (i.e., the name of the computer where SQL Server is installed). Additionally, if there are any security settings in the destination environment that effect the use of the default SQL Port of 1433 for the SQL Server, you may need to change this value accordingly. Once you have entered all the fields, click "Submit" and your new data resource will appear in the "list" below the "add" dialog box. Click on the "check" button to the left to "verify" the connection to the database. If all has gone well, a message will appear above that says "Connected to IBRD successfully". If you get an error message, clearly something needs to be corrected. This is probably one of the most difficult and/or sensitive portions of the installation and you may need to try several times to get a working connection. Although this is not what would be expected, it has been noted by first-hand experience that things seem to simply go better when you completely delete the data resource and start over, rather than just changing a couple of settings. At least, once you find settings that work, it is recommended that you write them down or save them in a text file, delete the resource, and do it "from scratch" one last time to ensure a "clean" installation of the resource.

2.3.9 - Test the new IBRD Server

You are now ready, finally, to make a local connection to the IBRD Web Site! Open a browser window (Internet Explorer recommended) and put in http://localhost:8101/ibrd/Login.jsp for the address. Note that the port number value of "8101" is assumed here, but should this actually not be the first JRun server installed on this system the assigned port might be different (e.g., 8102, 8103 etc.). It should also be noted that parts of this address are case sensitive, and care is needed (e.g., the "L" in Login.jsp is upper case). When the IBRD "Account Login" screen appears, put in the User Id of "sysmgr" with a password of "testibrd" and click the "Login" button (note: you must click the button, not just hit the Enter key). See the document titled "IBRD Getting Started User Manual" for a brief introduction to using the software. A list of initial user accounts for this Pilot test database are included there as Appendix A.

Step 2.4 – Configure DocumentManager.Properties

Back in step 2.2.1, several files were copied to the JRun servers folder (e.g., C:\JRun4\Servers). One of these files is named DocumentManager.Properties. As discussed in various sections above, this file contains settings that predominately affect the automatic generation of emails for the IBRD system. The critical changes that will be necessary in this file as part of the installation process are:

(1) If the recommendation of installing JRun4 at the root of the C: drive (i.e., C:\JRun4\) could not followed, all references in this file to "c:/jrun4/servers/IBRD" will need to be changed accordingly.
(2) The line with the keyword "EmailDispatcher.smtpHost" will need to be changed to reflect the appropriate reference for the local email server.
(3) Change the line with the keyword" EmailDispatcher.fromAddress" to contain the appropriate email address.

Note: values for email address and URL should also be set in SystemCfg.

Step 2.5 – Configure ProcessTwoYearRequest Batch Job and Scheduled Task

This requires two steps:

(1) Confirm that the field named "CONFIRMATION REQUEST TIME" in SystemCfg is set to the desired value (e.g., 60 days is a typical setting)
(2) Configure the program named "ProcessTwoYearRequest.bat" in the IBRD Server folder (e.g., "C:\JRun4\servers\IBRD") to run as a Scheduled Task on a once a day basis.

Step 2.6 – Configure Scheduled Tasks and OBDC for Archive Exes

This involves the followings steps:

(1) Configure the program named "IBRDFileArcPurge.exe" in the IBRD Server folder (e.g., "C:\JRun4\servers\IBRD") to run as a Scheduled Task on a once a day basis.
(2) Configure the program named "IBRDArcPurgeTables.exe" in the IBRD Server folder (e.g., "C:\JRun4\servers\IBRD") to run as a Scheduled Task on a once a day basis.
(3) Set up an Open Database Connection (ODBC) to the IBRD database for use by these two programs. Specifically, using the Windows utility to manage ODBC connections (under Administrative Tools in the Control Panel), create an entry with the name "IBRD" and a connection to appropriate SQL Server with the default database set to IBRD and employing SQL Authentification.

Step 2.7 – Configure IBRD JRun Server as a Windows Service

This is a critical step once everything is working correctly to ensure that when there is a hardware or operating system restart that the web site's JRun level support is restarted automatically as well. The following three steps create service named "JRun IBRD Server":

(1) Open an MSDOS Command Window
(2) Go the JRun4 "bin" folder (e.g., C:\JRun4\bin)
(3) Run the jrunsvc.exe application with the syntax: `jrunsvc -install IBRD`

Step 2.8 – Clear Tables and Text Logs

Assuming that testing is complete and the IBRD System is ready to go operational, the following tables in both the IBRD and the IBRDArchive databases should be cleared of all records:

Table Name	Comments
BeaconReportFact	Be aware of dependencies on various dimension tables (names end in "dim")
OperMsgLog	
LogUserAccess	
LogQueryAccess	
LogPrtEmailFax	
SarTransactionLog	
ConfigChangeLog	
Feedback	
RegistrationDb406	Specific arrangements to keep certain records for National Administrations could be arranged if appropriate.

Note: In all but RegistrationDb406 it is recommended that the Identity column be re-seeded to the value "1".

In addition all files in the following folders should be deleted:

Folder	Comments
C:\JRun4\logs	(actual path or drive letter may vary)
C:\JRun4\servers\IBRD\dispatch\temp	(actual path or drive letter may vary)

Step 2.9 – Set Up Initial Accounts

In effect the IBRD must be configured accordingly for the ultimate Runtime environment. This is a matter of setting up User accounts using the built in user interface capabilities of the application itself and/or direct population of the fields in the Users table. (See Section 5.1 of the IBRD System Maintenance Manual).

Step 2.10 – HTTP Server

Once the database, the application and the JRun configuration are all in place, some method of negotiating outside requests to and from the Internet is required. JRun provides such a "Web Server" capability. To date the Apache HTTP Server has been used in IBRD development and testing. Beyond observing that this capability is needed, this document goes no further in specifications as this will depend largely upon the environment in which the application is deployed.

- END OF SECTION 10 -

page left blank

ANNEX A

LIST OF ABBREVIATIONS AND ACRONYMS

COSPAS	Space system for the rescue of vessels in distress (Russian Federation)
COTS	Commercial off the shelf
C/S	Cospas-Sarsat
ELT	Emergency Locator Transmitter
EPIRB	Emergency Position Indicating Radio Beacon
FAX	Facsimile
FAQ	Frequently asked question
IBRD	International 406 MHz Beacon Registration Database
ICAO	International Civil Aviation Organization
ID	Identification
IMO	International Maritime Organization
IP	Internet protocol
ITU	International Telecommunication Union
MARS	Maritime mobile access and retrieval system (ITU database)
MCC	Cospas-Sarsat Mission Control Centre
MHz	Megahertz
MMSI	Maritime Mobile Station Identity
POC	Point of contact
PLB	Personal Locator Beacon
RCC	Rescue Coordination Centre
SAR	Search and Rescue
SARSAT	Search and Rescue Satellite Aided Tracking system (Canada, France, USA)
SBM	Shore-based maintenance
SQL	Structured query language
TAC	Type approval certificate
15 Hex ID	15 hexadecimal character identification

- END OF ANNEX A –

- END OF DOCUMENT -

page left blank

Cospas-Sarsat Secretariat
1250 René-Lévesque Blvd. West, Suite 4215, Montréal (Québec) H3B 4W8 Canada
Telephone: +1 514 500 7999 Fax: +1 514 500 7996
Email: *mail@cospas-sarsat.int*
Website: *http://www.cospas-sarsat.int*