Elastic COBOL Programmer's Guide

REVISION: SEPTEMBER 2014

Preface

This Programmer's Guide provides guidelines for the usage of Elastic COBOL programming constructs. Programming with Elastic COBOL adds features that extend standard COBOL-85 and offers functional capabilities which are not defined as part of the standard.

Elastic COBOL extensions should be easy to comprehend to the experienced COBOL programmer and a basic knowledge of COBOL-85 is assumed.

Trademarks

- IBM is a registered trademark of International Business Machines Inc.
- Oracle and Java are registered trademarks of Oracle and/or its affiliates.
- UNIX is a registered trademark licensed exclusively to X/Open Company Limited.
- Linux is a registered trademark of Linus Torvalds.
- Windows is a registered trademark of Microsoft Corporation.
- Eclipse is a trademark of the Eclipse Foundation Inc.
- Other names may be trademarks of their respective owners.

Product Usage

Development

Elastic COBOL application development is performed with the use of the Elastic COBOL compiler and runtime libraries. Elastic COBOL includes an Eclipse based Integrated Development Environment (IDE).

Development is permitted on development platforms where a valid development license has been purchased.

Deployment

Application deployment is permitted to platforms where a valid deployment license has been purchased.

Contents

Chapter 1 – COBOL Fundamentals

Elastic COBOL operates similar to many other COBOL compilers, but possesses certain proprietary execution requirements that may not be found with other traditional COBOL offerings. This section addresses these programming requirements.

Reference Formats

Elastic COBOL supports the standard fixed reference format that has expressed COBOL source code from the 1960's. Support also exists in Elastic COBOL for additional formats introduced over time by various COBOL vendors and includes the COBOL 2002 standard format extensions.

Fixed Format

```
[1 2 3 4 5 6] [7] [8 9 10 11] [12 ... 72] [73 ...]
 Sequence      I   Area A      Area B      Right Margin
```

Fixed is the traditional format, with sequence numbers in the leftmost six columns, an indicator column, areas A and B, and a right margin. Fixed format is assumed for most operations by default; however, the Elastic COBOL compiler will attempt auto-detection of other formats. The IDE may auto generate SOURCEFORMAT statements into the top of new source code fragments to force the compiler to handle the file in the desired manner.

Variable Format

```
[1 2 3 4 5 6] [7] [8 9 10 11] [12 ... 16383] [16384 ...]
 Sequence      I   Area A      Area B         Right Margin
```

Variable is similar to the traditional format, but without the right margin. This format was used primarily with one particular compiler, and should not be used in new code. Elastic COBOL does support it for maintaining existing code.

Free Format

```
[1] [2 ... 16383] [16384 ...]
 I   Area A & B    Right Margin
```

Free format removes the sequence area, overlaps the indicator column with the program area, and has no right margin. Free format is very similar to other modern languages in its formatting, allowing program code to appear everywhere with a liberal amount of white space. Inline comment indicators ('*>' and '|') were introduced to help with this format, as was nonnumeric literal concatenation ('&').

COBOL Files

Elastic COBOL expects COBOL source files to be line delimited text files, in the format appropriate to the system.

For most systems, this means ASCII formatted text files. For iSeries and zSeries systems, this means EBCDIC formatted text files. Elastic COBOL supports a smart auto detection system that is capable of compiling ASCII on EBCDIC machines and EBCDIC on ASCII machines, but the natural format for the system is recommended.

Elastic COBOL also supports Unicode COBOL source, where every character takes two bytes. It enters this mode automatically when the first two bytes of the file are a Unicode byte order marker.

The COBOL source files should not be record oriented unless the underlying operating system automatically converts from record oriented to text oriented files on demand.

Copy Files

Elastic COBOL expects line delimited text files as compiler input for both COBOL files and copy files. If the copy files are in a record oriented, tree, or another repository format, they must be converted to line delimited text files prior to invoking the Elastic COBOL compiler.

Users of the Elastic COBOL IDE can drag & drop any required copy files into the projects 'copylib' folder.

Users of the Elastic COBOL compiler from the command line should note that the compiler first searches the current directory, but also searches through the path specified by the environment variable COPYPATH for any copy files required by the COBOL program.

Specific file names that include the complete file extension must be placed within quotes. Elastic COBOL does not generally require quotes around the name, but they are required when including an extension such as COPY myfile.cpy; otherwise, the compiler will assume that the period defining the extension is the end of sentence. An unqualified copy file name will result in an error.

It is recommended to include the full filename rather than leave any extension implicit, but Elastic COBOL does support implicit copy file extensions of .cpy, .wcb, .cpb, .cdb, .cfd, .wrk, .lnk, .lks, .mnu, .prd, .evt, and .def. Elastic COBOL also supports the uppercase version of those extensions on systems supporting case-sensitive filenames.

The syntax *COPY x OF y* copies the filename x from the sub-directory y. So x must be a filename and y must be a directory name. This combination may be anywhere on the COPYPATH.

Literals

Literals are inline data items, such as "Hello" and 123.45.

Nonnumeric literals may be expressed using either single quotes ('Hello') or double quotes ("Hello"). Double quotes are standard and should be used in new code, but single quotes are supported for compatibility. No compiler options are necessary, and a single COBOL source code segment may contain both formats, but this practice is not recommended. A compiler option is available for handling quotes, but it only affects the meaning of the special register QUOTES.

Numeric literals may be expressed in base 10 normally, but also may be expressed in alternate base numerics. Normally the period (.) is used for fractional digits, but the comma (,) may be used with the expression DECIMAL IS COMMA clause.

Nonnumeric and numeric literals are normally expressed using a direct quote, but also may be expressed using based literals. Based literals specify that a literal is formed using an alternate numeric base, such as 16 or 8, rather than 10. They also specify that the literal is formed from the symbolic concatenation of digits rather than the actual text of the literal.

National literals may be expressed using either the N"literal" or G"literal" notation. It is recommended that the N"literal" notation be used in new code as it is the ANSI Standard method for defining National literals.

Chapter 2 – Program Lifecycle

Overview

The program lifecycle discussed in this guide is the COBOL program procedural flow and any data the program may access. The lifecycle is a runtime execution concept. There are three major components of the Elastic COBOL program lifecycle, not all are used in every program: *COBOL, Java and Native.*

COBOL may be simply by itself in a single program with no calls. This is true for small utility programs and helps to lead into the more complex applications.

COBOL calls to COBOL, when both are compiled in Elastic COBOL, are not significantly more complex than single COBOL programs. If the one COBOL program is compiled natively using a combination of COBOL compiler technologies, for and is discussed in this chapter as “COBOL to Native”.

COBOL calls to native code move the flow of control beyond the scope of the Java Virtual Machine. The native code may be written in a variety of languages, and since this moves beyond the platform independent nature of Elastic COBOL, the capabilities vary per platform. A 'C' language interface is discussed later as a mixed language application.

COBOL calls to Java are made through a variety of means, allowing either the COBOL side or the Java side to exist transparently of the other side.

Java calls to COBOL are enabled through the use of Elastic COBOL. To the Java programmer, COBOL programs compiled with Elastic COBOL are just like other Java classes.

Elastic COBOL programs can be separated into server and client components and run remote from one another. If one or more program units are in Java the program interface is treated in a similar manner to a program unit compiled with Elastic COBOL. A remote client or server in Java is a natural extension of the COBOL support within the Elastic COBOL compiler.

COBOL

A COBOL application is the most common form of program. The main COBOL program is the first or only one run in an execution session.

It begins at the first paragraph not defined in the declaratives, as do all COBOL programs. It ends at the final statement in the program or at a STOP RUN or GOBACK when there are no calling programs.

Any COBOL program may be a main program, so when there are multiple programs the system must be told which one is the main. Executing a program from a command line is done by specifying the program-id as part of the ‘java’ command. The COBOL program can be executed within the

Elastic COBOL IDE by simply running the selected program within the framework. *More information about using the IDE is available in the Elastic COBOL User's Guide.*

A COBOL program without a LINKAGE SECTION is naturally a main program. A COBOL program with a PROCEDURE DIVISION USING may be a main program in Elastic COBOL. It will retrieve its linkage arguments from the command line parameters automatically. This allows for easy unit testing of a subprogram, but this is not a portable construct.

Other COBOL vendors may use various methods for naming programs; some use the IDENTIFICATION DIVISION name and the filename name for identifying a program which results in two implicit names to every COBOL program. Elastic COBOL uses the IDENTIFICATION DIVISION name; this is specified via the PROGRAM-ID, CLASS-ID, etc. The name itself is externalized so it is valid for the file system and to reflect the case independent nature of COBOL. The externalized name has letters converted to lower case, and spaces to underscores. Certain reserved words may have an underscore appended. This externalized name is only necessary for the main when running from an external environment, so any CALL may include upper-case letters. When there are the two different names, the compiler will issue a warning when the two names do not match one another.

The program itself is compiled to a .class file, the externalized program name with the file extension .class. This is the Java executable. There are also .jar files, or Java Archives; these are an archive of .class files and other resources. In addition to this, .war, .ear and other formats exist which are specialized versions of .jar files.

The COBOL program is maintained in COBOL source code and executes within the Java execution environment. Elastic COBOL insulates the user from the need to learn Java programming concepts. However, like in the past, it is beneficial understand the application's operating system or execution environment, and so with Elastic COBOL, the user should be aware of some basic Java execution concepts.

The operating system uses a PATH environment variable to find executable programs. Java uses a CLASSPATH environment variable to find executable Java programs. Since a single CLASSPATH environment variable for the entire system is unwieldy, most executions use a CLASSPATH specified for the single execution to specify only the files necessary. The CLASSPATH must include the application program, the Elastic COBOL runtime, and any third-party or extension runtime like XML support. This is handled automatically from the Elastic COBOL IDE, but this knowledge can help with troubleshooting and deployment.

COBOL to COBOL

COBOL to COBOL is a call from one source COBOL program to another target COBOL program local to the machine. The target may be the same as the source for recursion. The four main mechanisms for transfer of control are CALL, INVOKE, THREAD and SESSION.

Call

The call must be made to the name given by the IDENTIFICATION DIVISION, not the source code's filename.

The call from source to target is done using the traditional COBOL CALL verb. The parameters may be any of the Elastic COBOL datatypes. The parameters may be passed by content, passing the actual data across, or by reference, passing a reference pointing back to the original data. Any data changes by the target are local to the target when passed by content, but is reflected back in the source program when passed by reference. Passing data by value picks BY REFERENCE or BY CONTENT as appropriate to the datatype for a call to Java or native code. Only traditional COBOL datatypes may be passed BY REFERENCE; OBJECT types may only be passed BY CONTENT or BY VALUE, but the compiler will automatically convert the REFERENCE to CONTENT type when necessary. Literals may only be passed BY CONTENT or BY VALUE, but the compiler will automatically convert the REFERENCE to CONTENT type when necessary.

If the call is not successful, the failure will be returned to the calling program. If an ON EXCEPTION clause is coded for the CALL, then the ON EXCEPTION handler has complete control. If no ON EXCEPTION clause is coded for the CALL, then a dialog will be presented to the user detailing the error. There are options for disabling this dialog under all circumstances. An extension to the ON EXCEPTION clause allows an identifier to be specified immediately after the keyword EXCEPTION; it will receive the exception text. As this exception text may grow without bounds, choose a PIC size suitable for display or diagnostics.

Control is transferred at the point of CALL, to return later to the same point under most circumstances. A STOP RUN in the target will end not only the target program but all other source programs in the call chain. Control may be returned from the target back to the source when the target issues a GOBACK or EXIT PROGRAM verb, or when the target reaches the end of its code.

The target program may be the same as the source program, either directly or indirectly through other intervening calls. This is recursion. As most data is shared between the source and target program, particular care must be taken with recursion.

To allow recursion to better operate, there is a LOCAL-STORAGE SECTION that operates in a similar way to the WORKING-STORAGE SECTION. Unlike WORKING-STORAGE that has only one instance per program session, the LOCAL-STORAGE is unique to each instance of the program, even within the same program session. The LOCAL-STORAGE section has a higher overhead per call in that it must be initialized each call, and it has additional memory per program instance.

In a call, COBOL programs have traditionally passed data back by reference. There is a more function oriented manner allowed as well by many COBOL implementations including Elastic COBOL. This involves coding RETURNING or GIVING identifier after the PROCEDURE DIVISION

USING. This value is returned to the caller in slot given by CALL RETURNING identifier.

Example 1:

```
77 INPUT-1 PIC 99.
77 INPUT-2 PIC 99.
77 OUTPUT-1 PIC 999.

PROCEDURE DIVISION USING INPUT-1 INPUT-2 RETURNING OUTPUT-1.
```

Example 2:

```
CALL "MY-FUNCTION" USING MY-INPUT-1 MY-INPUT-2 RETURNING MY-RESULT.
```

Invoke

The INVOKE verb is similar to the CALL verb, but is used for object oriented programming. It may INVOKE methods on objects written in either COBOL or Java; INVOKE on Java is covered later.

The COBOL program may define CLASS-ID style programs with METHOD-ID methods; this COBOL program may then be constructed and then have its methods invoked.

Control is returned at the end of the method unless a STOP RUN is called.

Thread

Threads are separate paths of control flow within the same process and session.

Certain threads are created automatically in the background by the runtime environment. Threads exist to paint the graphics on a graphical screen, print to the printer, or pre-parse XML.

Threading can also be under direct program control. The THREAD verb and THREAD options to other control flow verbs give direct control over program threading to the COBOL program.

THREAD operates in much the same way as GO TO, but instead of transferring control, a new thread is first created and then the GO TO is performed. The original thread of execution continues simultaneously with the new thread of execution. The threading is completely simultaneous only as allowed by multi-processor systems, but the illusion of simultaneity is maintained by the operating system.

PERFORM THREAD and CALL THREAD do the perform or the call in a separate thread, continuing with the original thread as well.

Most COBOL programs do not require threading, but it can be useful especially for long-running operations and can provide a better user experience.

Sessions

Sessions execute no different from threads in execution. The difference is the lifecycle of the associated data.

In a thread, data in the WORKING-STORAGE is shared between threads. This is fine for new programs that take explicit advantage of threading, but not for running multiple instances of traditional programs which were not constructed to take advantage of threading.

In a more traditional program, the program may be run as a whole multiple times on the same computer. Each copy of the program is running in an entirely separate process, the operating system's way of separating one program from another. A transactional program may have a copy for each transaction, with the operating system or other invisible layers handling this separation of concerns.

The session concept is a way of allowing that process model to exist within a single process, rather than requiring multiple processes. In this way, only a single startup is required, only a single copy of program code, yet multiple copies of the same program with different WORKING-STORAGE sections may exist simultaneously.

A session is not normally started from a COBOL program, but rather from the surrounding environment. When running as a Servlet or Enterprise JavaBean, Elastic COBOL programs are setup to automatically run as separate sessions with separate data. The session concept is important for calls from Java to COBOL and is covered more in depth there.

Sessions may be started from COBOL programs as well using the SESSION verb; it operates the same as the THREAD verb, but for the scope of its data.

WORKING-STORAGE is separate for each session of a COBOL program; no data sharing is available through WORKING-STORAGE. Rather, there is a SHARED-STORAGE section that is shared by all sessions in a single process. The SHARED-STORAGE should be used sparingly since sessions should not in general communicate with one another directly. Often, session environments will define their own mechanisms for inter-session communication; for example, Servlet sessions may use session variable parameters.

COBOL to COBOL Remote

Elastic COBOL supports the Remote Method Invocation protocol directly. A later generation beyond Remote Procedure Call, Elastic COBOL has adapted it to function with traditional COBOL programs. The call side is known as the client, the side with the remote program to call is known as the server.

A remote call is done the same way as a local call. The CALL name is different though; rather than a simple name, it must give a local and method of call.

The remote call name is "rmi:name@location". The rmi: specifies Remote Method Invocation, the method of the call. The name is the same IDENTIFICATION DIVISION name always specified. The @location specifies where to find the program, the machine on which it is located. For example, CALL "rmi:my-program@myhost.com". The parameters are always passed BY CONTENT, regardless of the convention requested. Data may be passed back from the call using the PROCEDURE DIVISION RETURNING identifier through to the CALL RETURNING identifier. This call is identical whether the code is COBOL or Java. The client side does not require anything else.

A simple name may be used in the program code and then be aliased to the complete name by setting a system property or configuration parameter. For example, the configuration file may contain "mysimplename=rmi:my-remote-program@myhost.com"; then a CALL "mysimplename" will actually refer to the remote program.

The setup for the remote program should be handled with care. The setup itself is not complex, but the program should be running only under an account with privileges only to the level necessary. Never run a remote program under Administrator or root accounts. If running a production remote program, follow all the security procedures normally associated with remote procedure calls.

The setup of the remote program is done in two parts, the registry and the program itself.

The registry is called 'rmiregistry' and is included with Java. It must be run first, as it is the registry service where the program will be registered. The registry acts a telephone book, a lookup service that knows what services are available on the system and how to connect remote callers to them. The rmiregistry program itself takes no parameters, but the CLASSPATH must include the application classes and the Elastic COBOL runtime (ecobol.jar). If the CLASSPATH is not set properly, the application will not be found. This is the most difficult part. The rmiregistry program will not return.

The program itself must be registered as a service on the server machine. Often, this is the same machine as the rmiregistry. To do so, again the CLASSPATH must be set to include the application and the Elastic COBOL runtime (ecobol.jar), but this may be done on the same command line. Run the command:

```
java -cp <ecobol. jar>;<application-dir> com.heirloomcomputing.ecs.run.rmi program-id…
```

Every program-id specified will be run as a service. The program-id is the externalized program-id name. Multiple program-id's may be run as services at the same time. After they report being registered and bound, they are ready to be called. This program will not return.

COBOL to Java

Elastic COBOL may call Java using a variety of methods. The burden of call conformance may be placed on the Java side or it may be placed on the Elastic COBOL side. When calling Java API functions, the burden is placed on the Elastic COBOL side. When calling Java classes made specifically for Elastic COBOL, the burden is placed on the Java side. In neither case is the burden heavy, but it influences design decisions.

Invoke

When calling API, Application Programming Interface, functions designed without regard to COBOL calling conventions, the Elastic COBOL program must conform to the calling convention expected by the API. The API will be documented for Java, so it may say that it is a static class with an int and a String parameter, or an object method with a short and a char. The INVOKE verb is used for this, and it is detailed extensively in the Language Reference INVOKE verb.

Use datatypes in COBOL matching the expected datatypes in the Java API. For byte, use SIGNED-BYTE; for short, use SIGNED-SHORT; for int, use SIGNED-INT; for long, use SIGNED-LONG; for String, use PIC X(n>1); for char, use PIC X(1). Always pass BY VALUE so it will conform to the type. Other type conversions are made implicitly when passing BY VALUE and these are documented at the INVOKE verb.

Callable interface

This section is for Java programmers wanting to grant direct access to Elastic COBOL programs using the CALL verb.

The most natural method for calling from COBOL to Java is the traditional CALL verb. This is useful especially for moving traditional COBOL programs to the Elastic COBOL environment to replace system functions with platform independent versions. The CALL verb can work with Java programs if they implement the *com.heirloomcomputing.ecs.api.Callable* interface.

The Callable interface must be implemented by any Java class to be called. It contains only one method to be defined, and it is the method which is called when the COBOL makes a call. The method has only two parameters, a Boolean array and an Object array; either or both of these may be null. It returns an Object or any descendent of Object. The method may throw an exception; it will invoke the ON EXCEPTION clause of the call to indicate call failure.

The Boolean array contains flags as to whether or not the call was requested as by reference or by content; by value will be converted to one of these automatically.

The Object array contains the actual data parameters passed. Usually, the elements of this array will implement

com.heirloomcomputing.ecs.api.Datatype for traditional COBOL datatypes, but the COBOL program may pass Object references explicitly as well. The number of parameters may be retrieved in the usual way, by examining the .length attribute of the array.

The Java class may be coded to allow any combination it desires. Check that the Object implements any necessary interfaces or extends any necessary class that the Java class may require; it is operating in the Java environment at this point, so the Java program has all the control it normally would. The Elastic COBOL application classes and methods are on the call stack, just as any other Java classes and methods would be.

Normally, the parameters implement *com.heirloomcomputing.ecs.api.Datatype*. Once cast to Datatype, they may be converted to or from any number of Java datatypes. The Datatype allows groups to be examined one element at a time, each of the elements being a Datatype as well. Datatype allows arrays to be examined one element at a time, each of the elements being a Datatype as well. The type of the Datatype may be examined to determine preferred actions. Any COBOL datatype that has a COBOL memory image (PIC X, PIC 9, etc.) will implement Datatype. Do not rely on any additional methods implemented by the actual class, rely only on the Datatype implementation.

The return type should be String, Number or Datatype. It will be treated as a move to the RETURNING identifier.

The following are the interfaces necessary for calling from Elastic COBOL to Java:

```
package com.heirloomcomputing.ecs.api;

/**
 * The Callable interface must be implemented by Java classes
 * in order to be called by COBOL programs using the CALL
 * verb.
 *
 * @see Datatype
 */
public interface Callable
{
    /**
     * This is the method called by the CALL verb for a program implementing
     * Callable.
     * <p>
     * If there are no parameters, byReference and params may be null.
     * <p>
     * If there are parameters, byReference.length and params.length
     * will be the number of parameters.
     * <p>
     * The Objects in params generally implement Datatype.  The Datatype
     * interface allows conversion to and from a variety of common Java
     * datatypes.
     * <p>
     * If the COBOL program passes Object declared as OBJECT [REFERENCE],
     * then the object itself will be passed directly.  If the Callable
     * is intended only for traditional data, just cast the data to
     * Datatype.
     * <p>
```

```
     * Throwing an Exception is allowed.  This will invoke the ON EXCEPTION
     * clause of the CALL.  If there is no ON EXCEPTION clause, then a message
     * dialog will appear with the information.
     * <p>
     * The COBOL program must be ready to handle the exception if user input
     * is to be suppressed from an exception.
     * <p>
     * Only use fromType methods on a param if the byReference is true.
     * <p>
     * @param byReference   possibly null, a Boolean for each parameter,
     *                      true if by reference, false if by content
     *                      (by value is generally by content)
     * @param params        possibly null, an Object for each parameter,
     *                      implementing Datatype for COBOL-oriented data,
     *                      though any object including null may be passed
     * @return value given to the CALL GIVING|RETURNING, should be String,
     *                      java.lang.Number (e.g., Integer) or Datatype.
     */
    public Object call(Boolean[] byReference,Object[] params) throws Throwable;
}

package com.heirloomcomputing.ecs.api;

/**
 * Datatype is an interface by which to describe data and data
 * conversions between Datatypes.  This interface is implemented
 * by COBOL-oriented data passed to Callable programs.
 *
 * @see Callable
 * @see Cancelable
 */
public interface Datatype
{
    // New Type Definitions (Bit Vectors)

    // All Invalid Types are negative.
    public static final int TYPE_INVALID=-1;

    // Masks
    public static final int TYPE_SIGN_MASK=7;
    public static final int TYPE_JUST_MASK=8;
    public static final int TYPE_NUMERIC_MASK=16;
    public static final int TYPE_CLASS_MASK=TYPE_NUMERIC_MASK|32|64;
    public static final int TYPE_VARIANT_MASK=128|256|512;

    // Sign
    public static final int TYPE_SIGN_NONE=0;
    public static final int TYPE_SIGN_LEAD=1;
    public static final int TYPE_SIGN_TRAIL=2;
    public static final int TYPE_SIGN_LEAD_SEP=3;
    public static final int TYPE_SIGN_TRAIL_SEP=4;

    // Justification
    public static final int TYPE_JUST_LEFT=0;
    public static final int TYPE_JUST_RIGHT=8;

    // Storage Class
    public static final int TYPE_NUMERIC=16;
    public static final int TYPE_NON_NUMERIC=0;

    // Storage Class Numerics
    public static final int TYPE_CLASS_ZONED=TYPE_NUMERIC|0;
    public static final int TYPE_CLASS_BINARY=TYPE_NUMERIC|32;
```

```
public static final int TYPE_CLASS_PACKED_DECIMAL=TYPE_NUMERIC|64;
public static final int TYPE_CLASS_FLOAT=TYPE_NUMERIC|32|64;

// Storage Class Non-Numerics
public static final int TYPE_CLASS_TEXT=TYPE_NON_NUMERIC|0;
public static final int TYPE_CLASS_GROUP=TYPE_NON_NUMERIC|32;
public static final int TYPE_CLASS_TABLE=TYPE_NON_NUMERIC|64;

// Storage Class Usage Formats
public static final int TYPE_VARIANT_0=0;
public static final int TYPE_VARIANT_1=128;
public static final int TYPE_VARIANT_2=    256;
public static final int TYPE_VARIANT_3=128|256;
public static final int TYPE_VARIANT_4=       512;
public static final int TYPE_VARIANT_5=128|    512;
public static final int TYPE_VARIANT_6=    256|512;
public static final int TYPE_VARIANT_7=128|256|512;

// Definition

/**
 * Return one of the TYPE_ types that best describes this Datatype.
 * The TYPE is a bit mask combination of the various TYPE_ constants
 * in this class.
 * <p>
 * The SIGN clauses describe the sign storage ability of the datatype.
 * The JUST clauses describe the left/right justification of the datatype.
 * The NUMERIC/NON-NUMERIC describe the general storage class; this overlaps
 * with CLASS.
 * <p>
 * The CLASS clauses describe the general numeric or non-numeric storage
 * format, but not the particular storage bytes used.  The particular
 * storage bytes used are further described with VARIANT clauses.
 * <p>
 * The GROUP and TABLE classes may be further analyzed using the
 * getElements() method to obtain the component Datatypes.
 * <p>
 * All invalid types are negative.
 *
 * @return TYPE_value
 */
public int getType();

/**
 * The length in internal bytes, where meaningful.
 *
 * @return length in bytes
 */
public int getLength();

/**
 * The number of decimal positions, where meaningful.
 *
 * @return number of decimal positions
 */
public int getDecimalPositions();

// Internal Access
//
// Internal Access does not conversion of the data; it is
// moved as directly as possible.

/**
```

```
 * Return copy of the internal byte representation.
 *
 * @return byte[] copy of the internal data.
 *
 * @see #fromByteArray
 */
public byte[] toByteArray();

/**
 * Return copy of the internal byte representation.
 *
 * @param param byte[] is the destination for the copy
 * @param offset into the byte[]
 * @param length for the given length
 */
public void toByteArray(byte[] param,int offset,int length);

/**
 * Set the internal byte representation using the given byte[].
 *
 * @param param the byte[] from which to copy.
 *
 * @see #toByteArray
 */
public void fromByteArray(byte[] param);

/**
 * Set the internal byte representation using the given byte[].
 *
 * @param param byte[] is the source of the copy
 * @param offset into the byte[]
 * @param length for the given length
 *
 * @see #toByteArray
 */
public void fromByteArray(byte[] param,int offset,int length);

// Conversion
//
// Conversion massages data if necessary as it is moved
// from one type to another.

/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into a Boolean.
 *
 * @return Boolean representation of Datatype.
 *
 * @see #fromBoolean
 */
public Boolean toBoolean();

/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into a byte.
 *
 * @return byte representation of Datatype.
 *
 * @see #fromByte
 */
public byte toByte();
```

```
/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into a char.
 *
 * @return char representation of Datatype.
 *
 * @see #fromChar
 */
public char toChar();

/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into a short.
 *
 * @return short representation of Datatype.
 *
 * @see #fromShort
 */
public short toShort();

/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into an int.
 *
 * @return int representation of Datatype.
 *
 * @see #fromInt
 */
public int toInt();

/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into a long.
 *
 * @return long representation of Datatype.
 *
 * @see #fromLong
 */
public long toLong();

/**
 * Conversion method.
 *
 * Return the conversion of the Datatype into a float.
 *
 * @return float representation of Datatype.
 *
 * @see #fromFloat
 */
public float toFloat();

/**
 * Conversion method.
 * Return the conversion of the Datatype into a double.
 * @return double representation of Datatype.
 * @see #fromDouble
 */
public double toDouble();
```

```
/**
 * Conversion method.    *
 * Return the conversion of the Datatype into a BigDecimal.
 * @return java.math.BigDecimal representation of Datatype.
 * @see #fromBigDecimal
 */
public java.math.BigDecimal toBigDecimal();

/**
 * Conversion method.  This method is here for symmetry
 * with fromDatatype; it generally will just return 'this'.
 * @return Datatype representation of Datatype.
 * @see #fromDatatype
 */
public Datatype toDatatype();

/**
 * Conversion method.
 *
 * A toText() method is used rather than toString() to allow
 * the this interface to be applied to existing classes with
 * existing toString() methods which may have different
 * meanings.
 *
 * The toText() method returns the human-readable text
 * representation of a Datatype, suitable for display to
 * the user.  This should convert any internal formats to
 * external formats and this should display international
 * characters correctly.
 * @return String representation of Datatype.
 * @see #fromText
 */
public String toText();

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the Boolean.
 * @param Boolean value by which to set the Datatype.
 * @see #toBoolean
 */
public void fromBoolean(Boolean param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the byte.
 * @param byte value by which to set the Datatype.
 * @see #toByte
 */
public void fromByte(byte param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the char.
 * @param char value by which to set the Datatype.
 * @see #toChar
 */
public void fromChar(char param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the short.
 * @param short value by which to set the Datatype.
```

```
 * @see #toShort
 */
public void fromShort(short param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the int.
 * @param int value by which to set the Datatype.
 * @see #toInt
 */
public void fromInt(int param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the long.
 * @param long value by which to set the Datatype.
 * @see #toLong
 */
public void fromLong(long param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the float.
 * @param float value by which to set the Datatype.
 * @see #toFloat
 */
public void fromFloat(float param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to be equal to the double.
 * @param double value by which to set the Datatype.
 * @see #toDouble
 */
public void fromDouble(double param);

/**
 * Conversion method.
 * Sets the contents of the Datatype to the BigDecimal.
 * @param java.math.BigDecimal value by which to set the Datatype.
 * @see #toBigDecimal
 */
public void fromBigDecimal(java.math.BigDecimal param);

/**
 * Conversion method.
 * Sets the contents of the Datatype using the contents
 * of the parameter Datatype.
 * @param Datatype value by which to set the Datatype.
 * @see #toDatatype
 */
public void fromDatatype(Datatype param);

/**
 * Conversion method.
 * Sets the contents of the Datatype from a Unicode String.
 * @param String value by which to set the Datatype.
 * @see #toText
 */
public void fromText(String param);

// Group/Array
```

```
    /**
     * If this Datatype is composed of other Datatypes,
     * return the Datatypes of which this is composed.
     * Those elements may in turn be composed of other
     * Datatypes.  This is useful for examing a group
     * structure by its elements rather than as a whole.
     * This must return null for Datatypes that cannot
     * be decomposed into other Datatypes.
     * <p>
     * This should not decompose into artificial Datatypes;
     * that is, a text Datatype should not return an array of
     * Datatypes for each character in the text under
     * normal circumstances.
     * <p>
     * The Object[] may contain other arrays rather than
     * Datatypes; these arrays for multi-dimensional
     * structures must be traversed to obtain all elements.
     *
     * @return Object[] elements in this Datatype.
     */
    public Object[] getElements();    // null if not array or group
}
```

COBOL to Java Remote

Refer to COBOL to COBOL Remote and COBOL to Java for most of the information regarding COBOL to Java Remote calls.

The COBOL program on the client is done the same way as if it were the COBOL client of a COBOL server. The name of the program is the Java class name.

The Java server is run the same as if it were a COBOL program, using *com.heirloomcomputing.ecs.run.rmi*, but use the Java class name instead of program-id. No stubs are necessary for standard RMI functionality; the stub for *com.heirloomcomputing.ecs.run.rmi* is already included in the Elastic COBOL runtime.

If using a custom RMI solution, there is an alternative method. Instead of running *com.heirloomcomputing.ecs.run.rmi* to serve the Java program, it may be served by implementing the RemoteCallable interface. In such a case, stubs need to be generated using rmic; the RemoteCallable interface must be exposed remotely and bound to the server. Follow the instructions for the custom RMI solution in this case.

The definition for RemoteCallable is:

```
package com.heirloomcomputing.ecs.api;

/**
 * The RemoteCallable interface implements a program callable using RMI.
 */
public interface RemoteCallable extends java.rmi.Remote
{
    /**
     * @param params The Serializable parameters to the function.
     */
    public java.io.Serializable call(java.io.Serializable[] params) throws Exception;
```

```
}
```

For RemoteCallable, it is important to note that all the parameters and the return code must be Serializable. This must be stressed, that all the parameters and the return code must be Serializable. The traditional COBOL datatypes are Serializable already, as are many of the basic Java types, but certain object references such as File are not.

Java to COBOL

Java code can call COBOL code as if it were just another Java class. The Java class for the COBOL program is based off the identification division, as discussed earlier.

Create an instance of the class using the default constructor; every instance of the class is a session. Every instance has its own WORKING-STORAGE section. Multiple instances of the same class may safely exist simultaneously. Each will retain its own state. Moreover, the calls each of the instances make to other COBOL programs will maintain their own state automatically, as if each instance where its separate process. All the standard COBOL rules about maintaining WORKING-STORAGE from one call to the next, the CANCEL verb, etc. are maintained correctly for the perspective of each separate session. This is what allows Servlets, Enterprise JavaBeans, and other advanced Java technologies to operate seamlessly with Elastic COBOL programs.

The COBOL object may be defined as either a CLASS-ID or as a PROGRAM-ID. If defined as CLASS-ID, then the methods available to it depend entirely upon the METHOD-ID's defined for the COBOL class. If defined as PROGRAM-ID, then there is normally only one method called on it, call. The COBOL program implements *com.heirloomcomputing.ecs.api.Callable*, just as a Java program being called by COBOL does.

Using this API, an array of Objects is passed to the COBOL program. The array, being created from Java, normally follows the Java conventions of supporting String and Number extensions (Integer, etc.), but custom implementers of *com.heirloomcomputing.ecs.api.Datatype* may be passed as well. If the PROCEDURE DIVISION USING has object parameters, then an object of the appropriate type may be safely passed. The PROCEDURE DIVISION RETURNING identifier is the result of the call.

In addition to this, the GET-PROPERTY and SET-PROPERTY phrases are available to use on data items. These generate getters and setters in the COBOL program, accessible from a Java program. The types are dependent upon the usage of the COBOL data, but the name used is a standard Java name. Getters begin with get, setters begin with set. Each hyphenated section of the variable name is distinguished by placing the first letter of the name in uppercase. This is a very simple way for COBOL and Java programs' data to interact.

Java to COBOL Remote

See COBOL to COBOL Remote for information on running the COBOL program as an RMI server.

See COBOL to Java Remote for the definition of RemoteCallable.

All COBOL programs are served using *com.heirloomcomputing.ecs.run.rmi* implement the RemoteCallable interface. The Elastic COBOL runtime (ecobol.jar) needs to be in the CLASSPATH of the client; the remote application itself does not. The remote COBOL program should be handled as if it only implements RemoteCallable.

Follow the normal RMI Java procedures for obtaining the remote reference and cast it to RemoteCallable. Then it can be treated much the same as a normal Java to COBOL call.

COBOL to Native

Elastic COBOL code may call native code, and does not require actual JNI (Java Native Interface) coding to do so. Whenever attempting to call native code, first be sure that the native function is actually necessary. There may be a COBOL function already included for the same job, or a small snippet of Java code may be able to replace the native function. When there is no alternative to native code, there are several methods available.

Programs

The easiest method for calling native code is used when the native code is an entire, executable program. The syntax for this is a simple extension, adding the keyword PROGRAM after CALL.

Example (Windows):

```
CALL PROGRAM "notepad.exe" USING "myfile.txt"
```

This calling functionality operates in as cross-platform a manner as possible. If the program is found using the normal procedures for the operating system, it is executed using the CALL parameters as command-line parameters.

In a similar method, an ASSIGN TO "|programname" creates an Inter-Process Communication (IPC) pipe. When the file is opened, the program is executed; the program's standard input/output channels are redirected to become the COBOL file, so input and output to the file become input and output to the program.

These call patterns do create a separate process; sometimes, this is desired, other times it is not. If a separate process is not desired, look further into these alternate methods.

Shared Libraries

In Windows and Linux, calls to shared libraries (.dll or .so extension) are very direct. This functionality is dependent upon certain capabilities of the architecture and operating system, so it is not available yet on all systems; for other systems, see the stub method below.

The shared library must be loaded, the functions of the library called, and then the library unloaded; the unloading is implicit at program termination if not done explicitly.

To load the library, CALL "library-name" (e.g., CALL "User32.dll"); the CALL will return a value for the library pointer. The value will be non-zero upon success; it will be zero upon failure. Until the library is unloaded, the functions in the shared library are available. To unload the library, CANCEL "library-name".

To call a function in the library, call it by its exported name, as expected. To pass parameters to a native function, the types must be matched carefully. If the native code expects a primitive type, pass by content, if the native code expects a pointer, pass by reference. The BY VALUE convention option picks the most appropriate form for most applications automatically.

The following table outlines the common C type to COBOL type correspondence. Elastic COBOL supports a number of synonyms for COMP-X or COMP-X-REV (depending on platform), useful for interacting with native or Java code.

C type	Bytes	COBOL Usage
char	1	SIGNED-BYTE, UNSIGNED-BYTE, PIC X(1)
short	2	SIGNED-SHORT
unsigned short	2	UNSIGNED-SHORT
int	4	SIGNED-INTEGER
unsigned int	4	UNSIGNED-INTEGER
long long	8	SIGNED-LONG
unsigned long long	8	UNSIGNED-LONG
float	4	COMP-1
double	8	COMP-2
char*	*	PIC X(n>1)
void*	*	PIC X(n>1), group

For this shared library functionality to work, the shared library name_native.dll or libname_native.so must be present on the system. These libraries are not required by Elastic COBOL for anything but native code support and may be safely omitted from redistribution when not required. The source code to these libraries is included with Elastic COBOL and is used for stub generation.

Example Win32Beep:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. WIN32BEEP.
```

```
 DATA DIVISION.
 WORKING-STORAGE SECTION.

*
* The type to pass to the API function.
*
 01 SoundType PIC X(4) COMP-X.
 01 ReturnValue PIC X(4) COMP-X.

*
* Some possible types of beeps.
*
 78 MB-ICONASTERISK PIC 999 VALUE 40.
 78 MB-ICONEXCLAMATION PIC 999 VALUE 30.
 78 MB-ICONHAND PIC 999 VALUE 10.
 78 MB-ICONQUESTION PIC 999 VALUE 20.
 78 MB-OK PIC 999 VALUE 0.
 78 MB-SPEAKER PIC S999 VALUE -1.

 PROCEDURE DIVISION.

*
* Load User32.DLL
*
* This step may be done instead by specifying the
* DLL-LINK=User32.dll program property; specifying
* the property will load a comma-delimited list of
* .DLL's before first accessing native code.  It
* will also automatically unload them at program
* exit.
*
 LOAD-DLL.
     DISPLAY "Loading DLL..." UPON SYSOUT

     CALL "User32.dll" GIVING ReturnValue

IF ReturnValue = 0 THEN
   DISPLAY "Could not load DLL, exiting." UPON SYSOUT
   STOP RUN
ELSE
   DISPLAY "DLL Loaded successfully." UPON SYSOUT
END-IF
.


 MESSAGE-BEEP.
     DISPLAY "Calling MessageBeep..." UPON SYSOUT

     MOVE MB-ICONASTERISK TO SoundType
*    MOVE MB-ICONEXCLAMATION TO SoundType
*    MOVE MB-ICONHAND TO SoundType
*    MOVE MB-ICONQUESTION TO SoundType
*    MOVE MB-OK TO SoundType
*    MOVE MB-SPEAKER TO SoundType

* BY CONTENT and BY REFERENCE are _very_ important
* for native calls; if SoundType were passed by
* reference (the default), then the address of
* its value would be passed instead of the value
* itself, which is not what the Win32 API function
* MessageBeep is expecting.  It's expecting a 4-byte
* integer, satisfied by the PIC X(4) COMP-X definition
* above.  Native calls automatically convert the
```

```
* data in COMP-X/BINARY/COMP-5 to the correct
* endian format.

    CALL "MessageBeep" USING BY CONTENT SoundType
                GIVING ReturnValue

    DISPLAY "Call complete (Status " ReturnValue ")"
          UPON SYSOUT

    IF ReturnValue = 0 THEN
   DISPLAY "MessageBeep Failed." UPON SYSOUT
ELSE
   DISPLAY "MessageBeep Succeeded." UPON SYSOUT
END-IF
.

* Unload the DLL.
*
* This step may be skipped if the DLL-LINK=User32.dll parameter
* is passed.

 UNLOAD-DLL.
    CANCEL "User32.dll"
    DISPLAY "Done." UPON SYSOUT
    .
```

Stubs

For all systems, C source code is included to the ecobol_native and ecobol_user shared libraries. This source is available so native code may be supported even on new systems. Not all systems support arbitrary calls to native libraries, so the native call support may be directly inserted into the source code for this library.

In this method, a few lines of code are added to check for the call, and then act on the call. Generally, the action for the call is linked in externally when forming the shared library.

The simplest method for supporting native code is to insert code into the sublen function in ecobol_user.c. The argument count, argument vector, length of each parameter, and by reference/by content status of each parameter is present. The ecobol_native library and ecobol_user library must both be compiled into shared libraries for the system; compiling ecobol_native may be skipped if already compiled for the platform. Other methods are available for compatibility with other implementations, and ecobol_native itself may be modified if desired.

The important point is that the final library must be sharable and follow any operating system specific constraints with regards to libraries capable of being load by the Java Native Interface.

Native to COBOL

The native library ecobol_call handles calling from native code to Elastic COBOL. The native code must be capable of loading a shared library for this to function; most native languages support this capability.

Source code to a C program calling an Elastic COBOL program, including parameter passing, is included as calltest.

Chapter 3 – User Interface

The user interface refers to the many ways a program may interact with the user. COBOL does not have a standard user interface definition methodology, and a variety of user interface mechanisms have been developed over time to address the requirement. Elastic COBOL has support for most direct and indirect COBOL program user interface possibilities.

Text

Line Oriented

The simplest text user interfaces are line oriented. The program may display one or more lines of text to the user, prompt for the user response, and continue based on the input accepted from the response. The process of displaying information and accepting the result is commonly called “DISPLAY/ACCEPT”. The default device for I/O in this way is the CONSOLE, a device appropriate for the application execution platform. For example, on Microsoft Windows and Linux with a graphical user interface this is a CONSOLE Window 80 characters by 24 lines.

Elastic COBOL support of the “DISPLAY/ACCEPT” text based line oriented interface can use SYSOUT, SYSERR and SYSIN devices. The program may DISPLAY "text" UPON SYSOUT to prompt the user. The program may then ACCEPT identifier FROM SYSIN to receive the user response. Error messages may be displayed by doing a DISPLAY "error" UPON SYSERR. The default of CONSOLE can be changed to SYSIN/SYSOUT with the compiler option **-run:system**.

While it is simple to write a program using the “DISPLAY/ACCEPT” method, and is highly portable, it is generally not considered the most desirable user interface for any but the simplest applications.

Screen Oriented

Elastic COBOL supports screen oriented input/output for text screens. Text screens may be graphical text screens that emulate a text environment under Windows, X/Windows System, or even within a browser. Text screens may be displayed upon an actual text terminal on systems with curses support.

Elastic COBOL will use a window created programmatically or will build one by default as the window is used. The program may perform a DISPLAY WINDOW verb to explicitly control the creation of the main window; this allows it to deviate from the green-on-black, 80x25 default settings. The DISPLAY WINDOW clause will function only on graphical systems, or on

text systems, it will activate the ON EXCEPTION clause of DISPLAY. The DISPLAY WINDOW ON EXCEPTION clause is the proper method of determining if the program is running on a graphical system. A non-graphical system will invoke ON EXCEPTION clause, a graphical system will invoke any NOT ON EXCEPTION clause.

The text screens may either be defined in the DATA DIVISION as a SCREEN SECTION, or they may be defined implicitly in the PROCEDURE DIVISION. A proper declarative definition in the SCREEN SECTION is the preferred method to be used; it allows for expansion to the graphical screen section format later and has better cursor control for the user.

The SCREEN SECTION defines records that correspond to displayable, acceptable screens. Multiple records may be displayed and accepted from the same visible screen. The record is displayed using DISPLAY record-name. The record is accepted using ACCEPT record-name.

When a record is displayed, it starts at the current screen position, the upper-left corner by default, unless a LINE or COLUMN clause is used. Each elementary item within the record is displayed in turn based on the order of definition. Each elementary item is displayed one after another unless positioning clauses intervention.

When a record is accepted, the acceptable fields within the record are accepted in turn, in order of definition. An acceptable field has a USING or TO clause referencing a target identifier. The cursor may be moved between fields using the TAB key or the mouse when present. Data entered into the fields are moved to the target identifiers automatically upon termination of the ACCEPT.

The text on a text screen may have its attributes set initially in its definition, or modified using the SET verb. Some common text attributes include the FOREGROUND-COLOR, BACKGROUND-COLOR, BOLD, etc.

The definitions of text input/output in the SCREEN SECTION may also generally appear in the DISPLAY and ACCEPT verbs; see the DISPLAY and ACCEPT verbs for more information. The use of “DISPLAY/ACCEPT” is generally discouraged since it does not maintain the separation of business logic from user interface, and does not allow the runtime to manage cursor control as extensively.

For more information about the options available within the screen section, see the language reference guide.

Graphics

Elastic COBOL offers a graphical extension to the SCREEN SECTION text syntax. The GRAPHICAL SCREEN SECTION is available on all graphical platforms supported by Elastic COBOL and not specifically restricted to a single platform as other graphical COBOL solutions.

Several graphical SCREEN SECTION samples are included with Elastic COBOL, including *person*, *dream home*, and *gfxsample*. These samples

demonstrate many of the concepts discussed here and illustrate many of the functions supported in Elastic COBOL.

The graphical screen section is still defined in the SCREEN SECTION declaratively or inline in the PROCEDURE DIVISION. The result of using the graphical screen section is a wide range of graphical components presenting and retrieving user information.

Graphical components include the following:

Component	Description
LABEL	Label for text, similar to a protected text field.
ENTRY-FIELD	Input field where user may enter data.
PUSH-BUTTON	Button which user may push (e.g., OK Button).
CHECK_BOX	Button which user may check on and off (Boolean).
RADIO-BUTTON	Grouped button from which user makes selection.
SCROLL-BAR	Scroll bar from which user may scroll to a value.
LIST-BOX	Box with a list of selectable items.
COMBO-BOX	Pull down list of items combined with entry-field.
FRAME	Graphical frame grouping items visibly for user.
TAB-CONTROL	Tabbed pane, allowing user to select tabs.
BAR	Graphical bar for drawing.
GRID	Grid for 2d data display and entry, similar to spreadsheet.
BITMAP	Image component, showing .BMS, .JPG, .GIF, .ICO graphics.
TREE-VIEW	Collapsible/expandable tree, showing hierarchy of data.
WEB-BROWSER	Simple HTML rendering component.
SLIDER	Similar to scroll-bar, with major and minor tick marks.
STATUS-BAR	Status bar for bottom of window.
MENU	Menu for top of window.

Each component has properties that apply to it, and most components share a number of properties. For example, most components have a FOREGROUND-COLOR property, but the radio-button has a GROUP property and the GRID has an X and Y property. The components and all their properties are described in the appendix. The component names are reserved words, the property names are reserved only within the context of the component.

The graphical screen section display is still done using DISPLAY, the graphical screen section accept is still done with ACCEPT. However, several additional verbs and verb formats are also available.

The MODIFY verb modifies properties of a graphical component; the properties may also be set in the DISPLAY of an inline component. The INQUIRE verb inquires the value of a property of a graphical component. Both MODIFY and INQUIRE are flexible in the data types expected and returned.

DESTROY eliminates a graphical component; the component may be later re-created with another DISPLAY. To retrieve memory used by complex components, the component use DESTROY.

DISPLAY will create the component the first time it is displayed automatically. Every time the same record is displayed, the properties are re-evaluated; this is very important to remember when the properties include cumulative effects, such as adding an item to a list. The effects of the

MODIFY act immediately, and the INQUIRE verb inquires from the current status of the component.

Graphics are defined in the SCREEN SECTION in much the same manner as the text screen section. The existing text screen clauses still function in the same manner, including LINE, COLUMN, FOREGROUND-COLOR, etc. The graphical screen section is defined in terms of character cells, so LINE and COLUMN still have the same meaning; but in addition to integer co-ordinates, fractional co-ordinates are also supported. The Elastic COBOL screen section is actually always graphical, smoothly integrating the existing text definitions into the more modern graphical definitions; while both text and graphics may be used simultaneously on the same visible screen, avoid doing so as the simultaneous mixture will appear to be awkward. Graphical components will always be shown over plain text.

Components also have action procedures of a few different types. A BEFORE PROCEDURE may be defined to act on a component before its use in an ACCEPT. An AFTER PROCEDURE may be defined to act on a component after its use in an ACCEPT. An EXCEPTION PROCEDURE may be defined to act on a component when an exception occurs in an ACCEPT.

An EVENT PROCEDURE may act any time an event occurs on a component. An EVENT should only be used where necessary, and event code should be small and quick to execute. Information about the event may be obtained by defined an EVENT STATUS identifier in the SPECIAL-NAMES. The EVENT-STATUS group item will be filled in with information relevant to the event that just occurred; its contents are not defined afterwards. The components document which events they issue in the appendix.

There is also a DISPLAY MESSAGE BOX verb that displays simple message dialog boxes to the user. This is a good mechanism for simple error messages to the user.

Converting from Text Screens to Graphical Screens

A text screen section may be converted to a graphical screen section through a number of fairly simple steps; the graphical screen section design enables a smooth migration path over time from the text screen section rather than a requirement for an abrupt change.

1. Add an explicit DISPLAY STANDARD WINDOW BACKGROUND-LOW statement allows the colors of the window to conform to those expected by the native windowing system.
2. Convert text output to LABEL components.
3. Convert text input to ENTRY-FIELD components. The FROM, TO, USING, and SECURE clauses are still valid.

4. Instead of a DISPLAY of spaces to clear the screen, DISPLAY BLANK SCREEN and destroy components explicitly using the DESTROY command.
5. Instead of accepting functions keys, add PUSH-BUTTON components with meaningful text and TERMINATION-VALUE=# where the # is the function key value expected. The CRT STATUS may then be used to obtain the value after the ACCEPT.
6. Add a MENU to the screen with termination values in much the same manner as the PUSH-BUTTON components.

A *txt2gui* sample included with Elastic COBOL and demonstrates these steps and further describes additional enhancements one at a time.

HTML

HTML is the user interface of choice for Servlets and CGI programs. The specifics in this section relate to HTML in the context of Elastic COBOL, not HTML in general. For specific HTML information, see any of the many HTML books or other resources available about this topic.

An HTML display is crafted from a tag that represents the markup for a page. The browser is free to interpret the display of this information. For example, the bold tag "<B>" may be interpreted as a bold font, color change, or even ignored completely. So, HTML output must generally be tested with multiple browsers to ensure its correct display.

The HTML format is very lightweight, not requiring significant overhead in sending the information across the web. There is generally no perceivable startup time for an HTML display, unlike an Applet, so the user response time can be good. The browser is a comfortable user interface for many users.

DISPLAY

When in a Servlet or CGI program, any DISPLAY UPON SERVLET-OUT output is sent to the browser. So, DISPLAY "<html>" UPON SERVLET-OUT sends the first line of an HTML page. When running in the Servlet environment, SERVLET-OUT is the default device for output rather than CONSOLE, so a plain DISPLAY "<html>" will also send to the browser.

DISPLAY UPON CONSOLE will still display to the graphical or text console of the server. This usage should be avoided in Servlets.

DISPLAY UPON SYSOUT and DISPLAY UPON SYSERR will send output to the sysout or syserr of the server. These output streams are generally redirected by the web server to a log file, so they may safely be used for such a purpose. The end user's browser will not see such messages, so never depend upon them for user interaction in a Servlet.

EXEC HTML

There is also an EXEC HTML. Any text in the program area between EXEC HTML and END-EXEC is sent to the browser, in the same manner that any text sent to a DISPLAY UPON SERVLET-OUT is done.

Whereas a DISPLAY may alternate freely between literal text and variables for its output, the EXEC HTML format is much more statically oriented. It may include variable output by using the Elastic COBOL recognized tag <hostvar name=variable-name></hostvar>; the <hostvar> tag will be replaced by the contents of variable-name dynamically at runtime, as if variable-name were to be displayed.

Both EXEC HTML and DISPLAY to the browser may be done within the same program and the same transaction. This allows EXEC HTML to include static content readily and DISPLAY to include dynamic content.

DDS

DDS, or Data Description Specification, is the method for user interface design originating primarily from the iSeries computers running OS/400. DDS definitions are characterized within an external text description file, COPY DDS- statements in the COBOL source code, and using virtual file input/output to do the presentation.

Elastic COBOL supports DDS when used in conjunction with the Elastic COBOL DDS Plug-in. The Elastic COBOL compiler will compile the DDS copy files and will execute the DDS displays in a platform independent manner. For more information on DDS for display, see the DDS Plug-in.

VPLUS

VPLUS is screen management for the HP 3000 series of computers. Elastic COBOL supports VPLUS currently only for the HP 3000 through the use of the original VPLUS; such screens are still tied to the original HP 3000 environment.

Native User Interface

Elastic COBOL may also allow the use of additional, native code user interfaces. These must use a CALL interface and interface using the native code capabilities of the platform. See the COBOL to Native section of the program lifecycle chapter for more information about native calls.

Localization

Resource Text Internationalization

Elastic COBOL supports resource text internationalization. This allows the one COBOL source program to refer to text by a symbolic name that is replaced at execution time by language- and region-sensitive text.

So, for example, a screen may have a label that represents a greeting. This greeting may be 'Good day' in English, 'Hi there' in American English, 'Guten Tag' in German, and 'Buenos Dias' in Spanish. The symbolic name is greeting and the actual text is language and region dependent.

There are three pieces to this internationalization: the locale, the resource text files and the program code.

In addition, the Elastic COBOL runtime itself allows internationalization of some of its pieces. The standard text resources are listed as well.

Locale

The locale is the combination of language and region. The operating system and Java runtime combination will generally determine the current locale automatically, according to information given during the operating system setup.

The locale identifier is the lowercase language identifier, then an underscore (_), then the uppercase region identifier. For example, English in the United States of America is en_US, the German language in Germany is de_DE (German language being Deutsch in Deutschland), and French in France is fr_FR.

The following program displays the current locale identifier:

```
identification division.
program-id. showlocale.

data division.
working-storage section.

77 the-language pic x(2).
77 the-region   pic x(2).

procedure division.
main-paragraph.

    accept the-language from configuration "user.language"
    accept the-region from configuration "user.region"

    display "Locale is " the-language "_" the-region upon sysout
    .
```

Resource Text Files

A resource text file is stored in text format, one line per item. Each item contains a line of name=value. The name is the symbolic name and the value is the locale dependent value. The symbolic name may include letters, digits and the period or full stop (.).

A number of resource text files may exist for the same program or run unit; the program must specify the base name for each. The base name is the name without locale information.

The runtime searches for resource text files in a specific order, based upon the current locale identifier. It will search for resource text by the most specific locale identifier to the least specific, general base name. In the following chart, the base name is represented by base, the language by la, the region by RE.

Resource Text File Search Order:

base_la_RE.properties

base_la_RE

base_la.properties

base_la.properties

base.properties

base

For example, the base file mytext running in English in the United States will search for:

mytext_en_US.properties

mytext_en_US

mytext_en.properties

mytext_en

mytext.properties

mytext

If the mytext file contains a reference to a label on the screen which represents a greeting, some mytext files which may exist would be:

```
mytext_en_US:
label.greeting=Hi There

my_text:
label.greeting=Good Day

my_text_de_DE:
label.greeting=Guten Tag
```

```
my_text_es:
label.greeting=Buenos Dias
```

This would make the label 'Hi There' for English speakers in the United States, 'Guten Tag' for German speakers in Germany, 'Buenos Dias' for Spanish speakers everywhere, and 'Good Day' for everyone else. The symbol name for each of these is 'label.greeting'; the name of the symbol must be the same in the resource text file and in the resource program code, but it is otherwise unimportant.

The resource text file is a program resource and it may be included in the deployment .jar file.

Resource Program Code

The program must be told the name of resource text files, and where to use their contents.

The program is told the resource text file location in the SPECIAL-NAMES, through a RESOURCE IS "base" clause. Any program using a resource should mention its usage, but it actually only has to be mentioned in a program before first use; the entire run unit has access to the same resources. To use the resource text file mytext, include:

```
SPECIAL-NAMES.
   RESOURCE IS "mytext"
   .
```

To use the contents of the resource text file, replace the direct text with a reference to the symbolic resource. For example, in the screen section when setting the title of a label to Good Day, it can be changed to reference a symbolic name 'label.greeting' instead.

```
05 LABEL TITLE = "Good Day".
```

Can be changed to:

```
05 LABEL TITLE = RESOURCE "label.greeting".
```

This change may be made wherever displayable text is recognized, including a plain DISPLAY. So, even:

```
DISPLAY "Good Day"
```

Can be changed to:

```
DISPLAY RESOURCE "label.greeting"
```

To be sure that some meaningful text is displayed even if the resource file is missing, the clause WHEN OMITTED "default-text" may be included after the RESOURCE "name". This default-text will never be displayed unless the

resource is missing entirely. So to have the same label with default text for when the resource is missing:

```
05 LABEL TITLE = RESOURCE "label.greeting"
   WHEN OMITTED "Good Day".
```

Be sure to allocate enough screen space for the different international resources in use. Not all languages will have the same number of characters for an item, so allow some additional space.

Standard Resource Text

The Elastic COBOL runtime itself allows for internationalization of some pieces. These pieces always have a default text, so they do not need to be specified, but they may be included to more fully internationalize the COBOL program. These standard pieces may be included in any resource text file; the symbolic name is the important piece.

Note that Symbolic names beginning with component are related to graphical screen section components. Calendar months and days are used only if Java itself does not already have a localization implementation for these values; in supported Java locales, these will already be defined.

Symbolic Name	Default Text
component.push-button.ok	&OK
component.push-button.cancel	&Cancel
component.message-box.yes	Yes
component.message-box.no	No
component.message-box.ok	OK
component.message-box.cancel	Cancel
component.calendar.month.1	Jan
component.calendar.month.2	Feb
component.calendar.month.3	Mar
component.calendar.month.4	Apr
component.calendar.month.5	May
component.calendar.month.6	Jun
component.calendar.month.7	Jul
component.calendar.month.8	Aug
component.calendar.month.9	Sep
component.calendar.month.10	Oct
component.calendar.month.11	Nov
component.calendar.month.12	Dec
component.calendar.day.1	Su
component.calendar.day.2	Mo
component.calendar.day.3	Tu
component.calendar.day.4	We
component.calendar.day.5	Th
component.calendar.day.6	Fr
component.calendar.day.7	Sa
component.calendar.month.previous	Previous Month
component.calendar.month.next	Next Month
component.calendar.year.previous	Previous Year
component.calendar.year.next	Next Year
component.entry-field.browse	Browse...
component.entry-field.digits	Digits
component.entry-field.dmy	DDMMYY
component.entry-field.mdy	MMDDYY

Symbolic Name	Default Text
component.entry-field.ymd	YYMMDD
component.entry-field.implied	Implied Decimal
component.entry-field.decimal	Decimal
component.entry-field.number	Number
component.entry-field.full-or-empty	Must Be Full or Empty
component.entry-field.required	Required Field
component.entry-field.native	Call to native language support intrinsic failed
component.entry-field.digits-only	This field can only contain digits (0-9)
component.entry-field.ymd-only	The field must be a valid date, in YMD order
component.entry-field.dmy-only	The field must be a valid date, in DMY order
component.entry-field.mdy-only	The field must be a valid date, in MDY order
dialog.error_calling	Error calling
dialog.abort_program	Abort Program?
dialog.enter_to_continue	Press Enter to continue:
dialog.enter_information	Enter information:
dialog.yes	Yes
dialog.no	No
dialog.ok	OK
dialog.cancel	Cancel
dialog.open	Open
dialog.close	Close
dialog.back	< Back
dialog.next	Next >
dialog.finish	Finish
dialog.apply	Apply
dialog.help	Help
dialog.stop	Stop
dialog.break	Break
dialog.browse	Browse...
dialog.exit	Exit
dialog.retry	Retry

Chapter 4 – Printing

Printing

Elastic COBOL printing is done through an Elastic COBOL print driver. The Elastic COBOL print driver passes printing information to lower levels of printing support. Elastic COBOL supports a native system print spooler, and, where applicable, local and remote printing along with other system printing capabilities.

Elastic COBOL currently includes two print drivers, one for JDK 1.1 and another for JDK 1.2. The JDK 1.1 printer driver has fewer capabilities and is not as fast as the JDK 1.2 printer driver. The font definitions for Java were changed between the versions, so it is best to target either JDK 1.1 or JDK 1.2+ for Elastic COBOL printing. Elastic COBOL's preferred target is JDK 1.2+ for printing support.

Before any printer can be used, it must be opened. It may be opened through a normal OPEN verb to a file assigned to a printer, or it may be opened using the *P$Open* call. The printer must be opened through an OPEN verb in order to use text printing in the print job.

The printer may be used for either text or graphical form printing.

When the print job is finished, the printer must be closed. This should be done using either the CLOSE verb or the P$Close call. Printers may buffer data this will force the buffer to close and the printer to complete the print job.

Assigning the Printer

To assign a file to a printer, use the ASSIGN clause like the printer was any other file. However, the filename is a virtual filename representing the printer device rather than a physical file on the computer's hard driver. (On some systems without direct printing support, the printer may be mapped to a device file on the computer's hard drive. This printer mapping will generally follow the normal standards for the system.)

The assignment may be done in the following format:

```
ASSIGN TO PRINT ["options"]          ; same as ASSIGN TO "line:printer:graphics/[options]"
ASSIGN TO PRINTER ["options"]              ; same as ASSIGN TO
"line:printer:graphics/[options]"
ASSIGN TO PRINTER-1 ["options"]    ; same as ASSIGN TO "line:printer:graphics/[options]"
ASSIGN TO "printer:[options]"
```

Options may be specified from the following, with slashes (/) as separators:

DRIVER=name	name is the classname of a custom printer driver, JDK11 or JDK12.
FONT=name	default font name
SIZE=#	default font size
COLS=#	columns (default printer record size)
ROWS=#	rows (default calculated according to page and columns)
CODEPAGE=name	name is Java encoding name, default is operating system default
ALIGNX=#	text only horizontal alignment in pixels (default none)
ALIGNY=#	text only vertical alignment in pixels (default none)
MARGINX=#	left margin (default dependent upon other sizes)
MARGINY=#	top margin (default dependent upon other sizes)
DPI=#	dots per inch (default 600)
BOLD	font is bold
ITALIC	font is italic
PLAIN	font is neither bold nor italic
GRAPHICS	enable graphics using tag <graphics>
GFX	enable graphics using tag <gfx>
TEXT	disable graphics
HEIGHT_ADJUST=#	default .97
PAGE_WIDTH_ADJUST=#	default .93
PAGE_HEIGHT_ADJUST=#	default .91
DIMENSION=#	default .825

Text

Text output is done through a series of writes to the printer file after it is opened. As all text is printed using graphical capabilities under current printer drivers, the text is rendered in a particular font.

The text font may be explicitly set during the open, but will default to a meaningful value dependent upon the print file's record size. An 80 column record will use a larger font than a 120 or 132 column record. This in turn implies a number of lines supported for the text output.

Text output automatically includes the carriage return and new line at the appropriate points when using the COBOL positioning clauses for the WRITE verb. Elastic COBOL does support inline new line control as well.

Text output will automatically eject the page when the end of page is reached.

Since text output figures are dependent upon the record size, font, printer and paper size, it is recommended to test the printer output against any desired printers.

Forms

Form printing is done through the P$ commands. These are CALL statements implemented within the Elastic COBOL runtime itself, available on all platforms. No new syntax is required to support this printing.

Form printing may be done on the same page as text printing or it may be done entirely separately. If opened using the P$Open command, then text printing is not available. Form printing also includes direct text printing where all information is specified.

The P$ commands use positioning co-ordinates for most operations. These may be specified in a variety of units, including 'I' for inches, 'M' for metric centimeters, 'C' for character units, 'T' or 'D' for device units, and 'P' for pixels. Default units may be set for the printer rather than specifying the units in each individual command.

The P$ commands allow inquiry as to certain printer or printer driver parameters. This allows the program to discover at runtime the attributes of a printer and change the output accordingly.

The P$ commands allow geometric shapes to be drawn directly using primitive drawing commands. It also allows a more general connector concept, allowing the printer driver to connect the dots with the appropriate line segments automatically. As connectors are added to the page, they are not rendered until the page is complete; then, only the appropriate connections are made between the connectors allowing for complex form generation.

The P$ commands themselves are fully documented in the call library appendix, but summarized here as to function.

Dialog Commands

P$CLEARDIALOG	Clear printer dialog values to default values.
P$DISABLEDIALOG	Disable the printer dialog from being displayed automatically to the user upon next open.
P$DISPLAYDIALOG	Display the printer dialog to the user. This is automatically done by an OPEN if enable dialog is true; this routine should generally not be called directly.
P$ENABLEDIALOG	Enable the printer dialog to be displayed to the user automatically upon next open. The default is that the printer dialog is enabled.
P$GETDIALOG	Get printer dialog attributes.
P$SETDIALOG	Set printer dialog attributes.

Drawing Commands

P$COMPONENTOUT	Output a graphical Component to the printer. This is generally used for outputting barcodes generated from C$BARCODE.
P$CONNECTOR	Place a connector on the current printer page. Connectors are suitable for creating lines forms, where line drawing art may have been used previously.
P$CONNECTORTHICKNESS	Set the thickness of connectors on the printed page.
P$DRAWBITMAP	Draw a bitmap image on the current printer page.
P$DRAWBOX	Draw a box or rectangle on the current printer page.
P$DRAWLINE	Draw a line on the current printer page between two co-ordinates.
P$DRAWOVAL	Draw an oval or circle on the current printer page.
P$GETPOSITION	Get the current printer drawing position from the printer driver.
P$LINETO	Draw a line from the current drawing position to the given co-ordinates on the current printer page.
P$MOVETO	Move the current printer drawing position to the given co-ordinates on the current printer page.
P$SETBOXSHADE	Set the printer shading color.
P$SETPAINTMODE	Set the current printer drawing mode.
P$SETPEN	Set the printer drawing pen's attributes.
P$SETPOSITION	Set the current printer drawing position.

Text Commands

P$CLEARFONT	Set the printer font back to the default font.
P$GETTEXTEXTENT	Get the dimensional extent of a piece of text, as rendered in the current font.
P$GETTEXTMETRICS	Get attributes of the current printer font.
P$GETTEXTPOSITION	Get the current printer text drawing position.
P$SETDEFAULTALIGNMENT	Set the default printer alignment for text.
P$SETFONT	Set the current printer font.
P$SETLINEEXTENDMODE	Set the vertical spacing to use when outputting a carriage-return (without linefeed) to the printer. This defaults to zero (0).
P$SETTABSTOPS	Set the printer tab positions.
P$SETTEXTCOLOR	Set the current printer text color.
P$SETTEXTPOSITION	Set the current printer text position.
P$TEXTOUT	Output text to the printer.

Drawing and Text Commands

P$SETDEFAULTMODE	Set the default printer mode, absolute or relative positioning.
P$SETDEFAULTUNITS	Set the default unit of printer measurement.
P$SETBOTTOMMARGIN	Set the printer bottom margin.
P$SETLEFTMARGIN	Set the printer left margin for this page.
P$SETTOPMARGIN	Set the printer top margin for succeeding pages.

Control Commands

P$CLOSE	Close the printer driver, automatically ejecting the

	current page.
P$COMMAND	The P$COMMAND passes an arbitrary command string to the COBOL printer driver. This is used for supporting unusual commands or commands that would not be available in most printer drivers.
P$DISABLEESCAPESEQUENCES	Disable escape sequences in text output.
P$EJECT	Eject the current page from the printer, feeding in the next page.
P$ENABLEESCAPESEQUENCES	Enable escape sequences in text output.
P$GETDEVICECAPABILITIES	Get printer device capabilities.
P$GETHANDLE	Get the handle number of the current printer.
P$OPEN	Open the printer driver, making a connection to the printer. This may automatically show a printer dialog box, depending on settings and version of Java.
P$SETHANDLE	Set the current printer to the given printer handle number.

Barcode

Elastic COBOL supports interfacing with third-part barcode components. These barcode components are useful mainly for printer output in Elastic COBOL, so they are covered here within the scope of printing. The third-party runtime library must be in the CLASSPATH at runtime in order to be found successfully.

As there is no standard Java interface to barcodes, a barcode driver must exist for the barcode to be used. Some barcode systems interface using fonts or other transparent mechanisms; if the barcode system works with Heirloom with no changes, then its usage is beyond the scope of this chapter.

A barcode driver is included with Elastic COBOL for Dragon Technology's JBarcode bean. The barcode implementation itself is separate from Elastic COBOL and must be purchased directly from the vendor The vendor information is available as of time of publication at the following URL: http://www.dragontechnology.com/barcode/

The barcode itself is generated as either an image or a graphical component using the c$Barcode command to interface with the barcode driver. This image or graphical component may then be output to the printer using the p$ComponentOut call command.

Generally, the c$Barcode command takes a number of properties to set, and then based upon those properties will return a rendered image or component suitable for output. The quality and content of the output is highly dependent upon the barcode implementation.

An example of generating a barcode for area code 95118 with visible text is:

```
77 graphical-object object reference.

Call "c$Barcode" using
  "value" "95118"
  "show-text-bool" "Y"
  "check-digit-bool" "Y"
  "angle-degrees-double" 90
  returning graphical-object
```

```
call "p$ComponentOut" using graphical-object 5 6
```

The barcode will be rendered using each set of two parameters as a property name and a value. The value of the barcode is '95118'; the text will be shown, a check digit included, and the text will be rendered at a 90 degree angle. This information is sufficient to render the barcode in this case. Other barcode drivers may require additional information.

The graphical-object variable is the container for the reference to the rendered component. After the call, it will either be NULL for an invalid value, or a valid reference. The CALL itself may fail, and that failure may be visible as a user dialog in this case; an ON EXCEPTION clause may be coded as for all calls to determine failure programmatically.

The p$ComponentOut command outputs the component image to the printer. It may be done multiple times for each component, if desired.

Advanced

Custom Elastic COBOL Printer Driver

Elastic COBOL supports printers through its own printer driver before being passed to the lower level. This allows additional features like text mode support and the P$ printer functions. The implementing classname may be specified as the DRIVER= parameter in the printer's assignment filename. All printing, text or form, will be routed through the printer driver. This interface could be implemented for plain text printers through custom serial or parallel ports, receipt printers, etc.

```
package com.heirloomcomputing.ecs.api;

/**
 * Generic Printing Functions
 *
 * Any class that implements this may be used as an Elastic COBOL printer.
*/
public interface GenericPrinter
{
    // P$DISPLAYDIALOG RETURN CODES
    public static final int PD_RETURN_OK=0;
    public static final int PD_RETURN_CANCELLED=1;
    public static final int PD_RETURN_ERROR=2;

    // Pitch Values
    public static final int PITCH_NORMAL_VALUE=0;
    public static final int PITCH_EXPANDED_VALUE=1;
    public static final int PITCH_COMPRESSED_VALUE=2;

    // Connectors

    public static final int CONNECT_LEFT=1;
    public static final int CONNECT_RIGHT=2;
    public static final int CONNECT_UP=4;
```

```
    public static final int CONNECT_DOWN=8;
    public static final int CONNECT_HORIZONTAL=CONNECT_LEFT|CONNECT_RIGHT;
    public static final int CONNECT_VERTICAL=CONNECT_UP|CONNECT_DOWN;
    public static final int CONNECT_ALL=CONNECT_HORIZONTAL|CONNECT_VERTICAL;

    public static final int CONNECT_SHADE=16;
    public static final int CONNECT_SINGLE=32;
    public static final int CONNECT_DOUBLE=64;
    public static final int CONNECT_ROUND=128;
    public static final int
CONNECT_TYPE=CONNECT_SINGLE|CONNECT_DOUBLE|CONNECT_ROUND;

    // ------
    // DIALOG
    // ------

    /**
     * Set the dialog to initial unset default values.
     */
    public void clearDialog();

    /**
     * Display Dialog Box to user.
     *
     * @return true for success, false for failure, RuntimeException for error
     */
    public Boolean displayDialog();

    /**
     * Get Dialog attributes.
     *
     * @param attribute the Dialog attribute.
     * @return value of attribute
     */
    public String getDialog(String attribute);

    /**
     * Set Dialog attributes.
     *
     * @param attribute The Dialog attribute.
     * @param value The value of the attribute.
     */
    public void setDialog(String attribute,String value);

    /**
     * Return a component related to the printing; this is suitable ONLY
     * for usage with obtaining an image, and it is NOT guaranteed to
     * be anything other than null for a COBOLPrinter, especially on
     * later edition servers.
     */
    public Component getComponent();

    // -------
    // DRAWING
    // -------

    /**
     * @return the scale in horizontal pixels for the unit name.
     */
    public double getScaleX(String unitName);

    /**
     * @return the scale in vertical pixels for the unit name.
```

```
     */
    public double getScaleY(String unitName);

    /**
     * Draw a Bitmap.
     */
    public void draw(Image image,double xpos,double ypos,double width,double height);

    /**
     * Draw a Component
     */
    public void draw(Component component,double xpos,double ypos,double width,double
height);

    /**
     * Draw a Box, shaded or 3d.
     */
    public void drawBox(double xpos,double ypos,double width,double height,boolean
shade,boolean _3d,boolean raised);

    /**
     * Draw a Rounded Box, optionally shaded.
     */
    public void drawBox(double xpos,double ypos,double width,double height,boolean
shade,int arcWidth,int arcHeight);

    /**
     * Draw an Oval, shaded or 3d.
     */
    public void drawOval(double xpos,double ypos,double width,double height,boolean shade);

    /**
     * Draw a partial Oval, optionally shaded.
     */
    public void drawArc(double xpos,double ypos,double width,double height,boolean
shade,int startAngle,int endAngle);

    /**
     * Draw a line; set the next from to the current to.
     */
    public void drawLine(double x1,double y1,double x2,double y2);

    /**
     * Set the initial drawing location for the next drawLineTo; does
     * not render anything.
     */
    public void drawLineFrom(double x1,double y1);

    /**
     * Draw a line to the given coordinates, from the last drawn line.
     */
    public void drawLineTo(double x1,double y1);

    /**
     * Set shading color.
     */
    public void setShadeColor(Color shade);

    /**
     * Get shading color;
     */
    public Color getShadeColor();
```

```
    /**
     * Set the pen attributes.
     *
     * @param penStyle; from PEN_STYLE_SOLID (0), PEN_STYLE_DASH (1),
PEN_STYLE_DOT (2), PEN_STYLE_DASH_DOT (3), PEN_STYLE_DASH_DOT_DOT (4),
PEN_STYLE_NULL (5)
     */
    public void setPen(int penStyle,double penWidth,Color penColor);

    /**
     * @return ending X position of previous print
     */
    public double getPositionX();

    /**
     * @return ending Y position of previous print
     */
    public double getPositionY();

    /**
     * @param set position X of next print
     */
    public void setPositionX(double xpos);

    /**
     * @param set position Y of next print
     */
    public void setPositionY(double ypos);

    // -----------------
    // BASIC PRINTER OPS
    // -----------------

    /**
     * Eject the current page.
     */
    public void eject();

    /**
     * Open the printer according to filename.
     *
     * @param rawMode true if should open in rawMode, false if cooked
     * @param filename
     * @return true for success, false for failure (cancelled by user)
     */
    public Boolean open(Boolean enableDialog,boolean rawMode,String filename);

    /**
     * Close the printer access.
     */
    public void close();

    // -----------------
    // TEXT MANIPULATION
    // -----------------

    /**
     * Clear the font to default.
     */
    public void clearFont();

    /**
     * @param bottom is false if top of text, true if bottom
```

```
     * @return ending X position of previous print
     */
    public double getTextPositionX(Boolean bottom);

    /**
     * @param bottom is false if top of text, true if bottom
     * @return ending Y position of previous print
     */
    public double getTextPositionY(Boolean bottom);

    /**
     * Set the current font.
     *
     * @param f current font.
     */
    public void setFont(Font f,boolean underline,boolean strikeOut,boolean doubleStrike);
    public Boolean isFontUnderline();
    public Boolean isFontStrikeOut();

    /**
     * Get the current font.
     */
    public Font getFont();

    /**
     * Get the current font metrics.
     */
    public FontMetrics getFontMetrics();

    /**
     * Set the amount of space between two overwritten lines.
     */
    public void setLineExtend(double extend);

    /**
     * Set the pitch of the font.
     */
    public void setPitch(int pitchType);

    /**
     * Set the tab stop increment.
     */
    public void setTabStop(double tabStops);

    /**
     * Get the text color.
     */
    public Color getTextColor();

    /**
     * Set the text color.
     */
    public void setTextColor(Color textColor);

    /**
     * Set the text position x.
     *
     * @param bottom is true if bottom, false if top
     */
    public void setTextPositionX(double xpos,boolean bottom);

    /**
     * Set the text position y.
```

```
     *
     * @param bottom is true if bottom, false if top
     */
    public void setTextPositionY(double ypos,boolean bottom);

    /**
     * Output text at position, with a box or shade.
     */
    public void textOut(String text,double xpos,double ypos,boolean box,boolean shade);

    /**
     * Write plain text, with possible escapes if enabled; this MAY cause
     * a page eject and movement to the next page.
     *
     * @return true if page ejected
     */
    public Boolean write(String text);

    /**
     * Get renderable text extend in width and height as Dimension.
     *
     * @return physical dimension of text
     */
    public Dimension getTextExtent(String text);

    /**
     * Set the left margin.
     */
    public void setLeftMargin(double leftMargin);

    /**
     * Get the left margin.
     */
    public double getLeftMargin();

    /**
     * Set the top margin.
     */
    public void setTopMargin(double topMargin);

    /**
     * Get the top margin.
     */
    public double getTopMargin();

    /**
     * Set the bottom margin.
     */
    public void setBottomMargin(double bottomMargin);

    /**
     * Get the bottom margin.
     */
    public double getBottomMargin();

    // ---------------
    // PRINTER CONTROL
    // ---------------

    /**
     * Set device modes. (ChangeDeviceMode)
     */
    public void setDeviceMode(String attrib,String value);
```

```
    /**
     * Get the value of a device attribute.
     *
     * @return value of attrib
     */
    public String getDeviceCapability(String attrib);

    /**
     * Enable or disable escape sequences.
     *
     * @state true if escape if sequences are to be enabled.
     */
    public void setEscapeSequences(Boolean state);

    /**
     * Get printer info attribute.
     *
     * @return value of attribute.
     */
    public String getPrinterInfo(String attrib);

    // Connectors

    /**
     * Set the connector thickness, in pixels away from the center.
     */
    public void setConnectorThickness(int thickness);

    /**
     * Set a connector of the given type on the page; the page
     * is responsible for connecting connectors appropriately.
     */
    public void setConnector(double xpos,double ypos,int type);

    /**
     * XORMode
     *
     * This painting mode for drawing to the printer has the attribute
     * such that drawing the object twice will return the graphics to
     * the original state.
     */
    public void setXORMode(Color c);

    /**
     * PaintMode
     *
     * This is the normal painting mode for drawing to the printer.
     */
    public void setPaintMode();

    /**
     * Command
     *
     * Send arbitrary text commands to the Elastic COBOL printer driver; this may
     * be interpreted in any manner by the Elastic COBOL printer driver.
     */
    public String command(String command);
}
```

Custom Elastic COBOL Barcode Driver

Elastic COBOL supports a generic barcode architecture, allowing different Barcode drivers to be used from the standard "C$Barcode" generation function. A custom barcode driver may be selected by specifying the "DRIVER" property and the classname of the custom driver. Any barcode driver must implement the following interface:

```
package com.heirloomcomputing.ecs.api;

/**
 * Generic Barcode Functions
 *
 * This class must be implemented by any Barcode implementation to
 * be usable from C$BARCODE.
 *
 * The method of usage from C$BARCODE is:
 *  Construct the implementation..
 *  Set the properties.
 *  Call getComponent(), returning the barcode component.
 */
public interface GenericBarcode
{
    /**
     * The DRIVER property is never passed in setProperty; rather,
     * it is the property that specifies the implementation of
     * the GenericBarcode interface.
     */
    public static final String PROPERTY_DRIVER="DRIVER";

    /**
     * Wherever there is a match to one of the listed types, recognize
     * the listed type in addition to any other proprietary type names
     * supported.
     *
     * Code128
     * Code39
     * Code39_2to1
     * ExtendedCode39
     * ExtendedCode39_2to1
     * Interleaved25
     * Interleaved25_2to1
     * Codebar
     * Codebar_2to1
     * MSI
     */
    public static final String PROPERTY_TYPE="TYPE";

    /**
     * The value or code of the barcode.
     */
    public static final String PROPERTY_VALUE="VALUE";

    /**
     * Whether or not a check digit should be included.
     */
    public static final String PROPERTY_CHECK_DIGIT_BOOL="CHECK_DIGIT_BOOL";

    /**
     * Whether or not the text of the barcode value should be included.
```

```
     */
    public static final String PROPERTY_SHOW_TEXT_BOOL="SHOW_TEXT_BOOL";

    /**
     * The narrowest bar width.
     */
    public static final String
PROPERTY_NARROWEST_BAR_WIDTH_INT="NARROWEST_BAR_WIDTH_INT";

    /**
     * The angle of the barcode printing in degrees.
     */
    public static final String
PROPERTY_ANGLE_DEGREES_DOUBLE="ANGLE_DEGREES_DOUBLE";

    /**
     * The height of the barcode.
     */
    public static final String
PROPERTY_BARCODE_HEIGHT_INT="BARCODE_HEIGHT_INT";

    /**
     * The background color of the barcode.
     */
    public static final String
PROPERTY_BACKGROUND_COLOR="BACKGROUND_COLOR";

    /**
     * The background color within the barcode.
     */
    public static final String
PROPERTY_BARCODE_BACKGROUND_COLOR="BARCODE_BACKGROUND_COLOR";

    /**
     * Set a property on the barcode.
     */
    public void setProperty(String property,Object value);

    /**
     * Return the final barcode object, reflecting properties as
     * set by setProperty.
     *
     * This will generally be either a Component or an Image.
     */
    public Object getObject();
}
```

Chapter 5 – Data Access

Datatype Storage

Datatype storage is the manner in which datatypes are stored to memory, and affects the PICTURE and the USAGE of the variable.

This is not important for all programs, but can be important when redefining across memory and when storing the memory to permanent storage (such as in the file section). It can also be important when passing data to native code programs that cannot automatically determine the storage format used.

Because of the differences in data storage compatibility between vendors, when accessing data it is important that the datatype storage conventions match wherever there is an observable impact.

Some datatype conventions, such as the AcuCOBOL convention, have COMP-1 and COMP-2 with different general categories than most other COBOL compilers, so it can be very important to have a good match of data. Fortunately, Elastic COBOL supports most datatype conventions and has options to support various combinations. The USAGE clause documentation lists all available USAGE combinations.

When compiling code originally written for Micro Focus, compile using the Micro Focus data compatibility option. When compiling code originally written for AcuCOBOL, compile using the AcuCOBOL data compatibility option, etc.

Character Datatypes

The character datatypes represent data non-numeric in form, generally expressed as PIC X(n).

Character data has the simplest storage. The character 'A' is stored as an 'A', the character 'B' is stored as a 'B', etc. ASCII is the most common storage format, but Unicode and other encoding are also available.

The data is stored the same on all platforms and is the safest way to pass data from one COBOL implementation to another.

Fixed Point Numeric Datatypes

The fixed-point numeric datatypes represent data numeric in form, generally expressed as PIC 9(n) or PIC 9(n)V9(m), with a fixed number of whole and fractional digits.

The internal storage of fixed-point numeric differs considerably according to usage, data compatibility, and even platform. As COBOL did not standardize the internal storage of data, the different COBOL vendors do not

have compatible storage implementations. Elastic COBOL accommodates this through the usage of compiler switches (data settings in the IDE).

The various large categories of fixed-point numeric are the following:

Zoned Decimal

Binary

Packed Decimal

Zoned Decimal

Zoned Decimal is the simplest of the numeric storage formats, where each digit of the number occupies a byte of memory. The implied decimal point is only implied, not stored, so it occupies no memory. Generally, the sign is overlaid with the trailing digit, but the sign storage clause can modify this behavior.

The positive overlaid digit, the negative overlaid digit, and the unsigned digits of the numeric body may have different storage values. For many of the data compatibilities, the positive overlaid digit and the unsigned digit have the same storage. Sometimes, other data compatibility switches may have been used for compiling elsewhere, so knowledge of these data formats may be useful.

When stored with the sign separate, USAGE DISPLAY is the most standard way of passing data from one COBOL implementation to another.

In RM/COBOL compatibility, COMP is COMP-D-2 compatible.

Elastic COBOL, Micro Focus, IBM ASCII data compatibility

	0	1	2	3	4	5	6	7	8	9
Unsigned	0	1	2	3	4	5	6	7	8	9
Positive	0	1	2	3	4	5	6	7	8	9
Negative	p	q	r	s	t	u	v	w	x	y

AcuCOBOL data compatibility

	0	1	2	3	4	5	6	7	8	9
Unsigned	0	1	2	3	4	5	6	7	8	9
Positive	0	1	2	3	4	5	6	7	8	9
Negative	I	J	K	L	M	N	O	P	Q	R

HP COBOL-II, RM/COBOL data compatibility

	0	1	2	3	4	5	6	7	8	9
Unsigned	0	1	2	3	4	5	6	7	8	9
Positive	{	A	B	C	D	E	F	G	H	I
Negative	}	J	K	L	M	N	O	P	Q	R

IBM EBCDIC data compatibility

	0	1	2	3	4	5	6	7	8	9
Unsigned	xf0	xf1	xf2	xf3	xf4	xf5	xf6	xf7	xf8	xf9
Positive	xc0	xc1	xc2	xc3	xc4	xc5	xc6	xc7	xc8	xc9
Negative	xd0	xd1	xd2	xd3	xd4	xd5	xd6	xd7	xd8	xd9

COMP-D data compatibility (sign separate)

	0	1	2	3	4	5	6	7	8	9
ALL	x00	x01	x02	x03	x04	x05	x06	x07	x08	x09

COMP-D-2 data compatibility (sign separate)

	0	1	2	3	4	5	6	7	8	9
ALL	x00	x01	x02	x03	x04	x05	x06	x07	x08	x09

Binary

Binary numeric storage uses base two storage to store the value in the minimum space available. Size error detection may occur at either the maximum storage capability or the picture; it defaults to the picture representation, but may be modified using the -dt:truncbin compiler option.

Binary may be stored in big-endian or little-endian format. The storage format for some usage is fixed, for others it depends upon the platform. BINARY by default is big-endian. BINARY-REV by default is little-endian.

In Micro Focus, HP, and AcuCOBOL modes, COMP is a BINARY type.

In AcuCOBOL and RM/COBOL data compatibility modes, COMP-5 and COMP-N have opposite meanings on little-endian platforms such as Windows; in Micro Focus, COMP-5 has the opposite meaning on little-endian platforms such as Windows.

COMP-4 by default is platform independent BINARY, same as BINARY. COMP-5 is a platform dependent binary in some data compatibility modes; in Elastic COBOL data compatibility, it is the same as BINARY.

COMP-X allows its binary usage to be defined in terms of the minimum space to hold the number of PIC 9's specified, or the number of bytes expressed by PIC X's. COMP-X is platform independent ordering. COMP-N is like COMP-X, but platform dependent ordering in some data compatibility modes; in Elastic COBOL data compatibility, it is the same as COMP-X.

COMP-S is always a short, two byte binary format.

The binary types are good especially for data transfer with Java and native applications in C.

Packed Decimal

Packed Decimal is a base 10 storage format, where two digits occupy one byte, one nibble or half a byte per digit. On average, half as many bytes are required for packed decimal storage as for zoned decimal storage. The packed decimal usage vary in storing padding bytes, order of the nibbles within the byte, and whether or not unsigned numbers will still possess a sign nibble area.

When stored, the sign nibble is always separate from the numeric digit nibble, never overlaid.

COMP is a packed decimal storage format in Elastic COBOL data compatibility.

COMP-6 is a packed decimal storage format that stores sign only when necessary.

Floating Point Numeric Datatypes

The floating-point numeric datatypes represent data numeric in form, but without a fixed number of whole and fractional digits.

Elastic COBOL stores its floating point numbers in IEEE format, or the reverse byte order version of IEEE. COMP-1 and COMP-2 are single and double precision IEEE, while COMP-1-REV and COMP-2-REV are single and double precision reversed IEEE.

COMP-1-MVS and COMP-2-MVS are available for storing floating point numbers in the original MVS floating point format. These types are most useful for data transfer with old mainframe data and should not be used for other purposes.

Floating-point numbers are not precise and should never be used for storing monetary data. They are used primarily for passing data to floating point types in Java or native C code, and for scientific calculations. They may be used for approximating functions.

File Storage

For all files, the data stored in the file is dependent upon the storage of data in memory. The file commands take the internal storage of the data and make it persistent, storing it to disk, or return it from disk to the internal storage of memory. So, when attempting to read or write an existing file, first be sure that the datatype format matches.

Where the file format differs between COBOL implementations, Elastic COBOL supports the concept of a file protocol to specify how to access the file. The file may be specified as a Micro Focus compatible file, an AcuCOBOL compatible file, etc. This file protocol is specified using the protocol name then a colon then the filename itself. For example, assigning to a Micro Focus file called myfile would be an assign to "mf:myfile".

As certain vendors support multiple filesystems, Elastic COBOL does not necessarily support all filesystems. Check for each file type to see which types are supported.

Filenames on different platforms have different default directory separators; the separator character on Windows is the backslash (\) character, while in Unix it is the slash (/) character. The SPECIAL-NAMES allows the implied directory separator to be specified using the following syntax:

"x" IS FILE CHARACTER

Whenever the character x is found in a filename, it is replaced with the actual directory separator for the platform. This is used most to make explicit the implicit directory separator character used in the program code assignments. For example, when a typical assign is ASSIGN TO "mydir/myfile", then "/" IS FILE CHARACTER would make explicit that the '/' character is the separator; then it would be replaced by a '\' character in Windows automatically.

Sequential

Sequential files are declared as ORGANIZATION SEQUENTIAL. This is the default for files, and so is the organization used when no organization is explicitly specified. Sequential files may only be read, updated, and written in order. There is no random access to the file allowed.

Sequential files may either be fixed length or variable length. Fixed length files are the result of file definitions where there is only one record, or multiple records of the same length. Variable length files are the result of file definitions where multiple records are defined, not all of the same length.

Fixed length sequential files have no additional header information or meta-information within the file. Because of this, they are the same between most COBOL vendors and are well suited to data transfer between COBOL implementations.

Variable length sequential files may have header or meta-information within the file. This is to indicate the length of each record. Because of this, not all COBOL implementations store variable length sequential files in the exact same manner.

Elastic COBOL supports its own format, the Micro Focus sequential file format using the "mf:" protocol, and the AcuCOBOL sequential file format using the "acu:" protocol.

In addition, the RM/COBOL fixed length sequential file format is supported as its format is the same as the Elastic COBOL format. RM/COBOL variable length sequential file format is not supported.

Sequential and line sequential files may have protocols beyond simple storage. These additional protocols may specify virtual devices rather than actual files on the system. This allows items such as the system clipboard, or TCP/IP sockets to be used as files. These file protocols are documented in the language reference section on I/O.

Line Sequential

Line sequential files are declared as ORGANIZATION LINE SEQUENTIAL. This is for text file support, and matches the text file support expected by the system complete with line separators to separate each line. All line sequential files are implicitly variable length.

Whitespace at the end of lines is removed upon writing, and implied when reading.

As this corresponds to a text file, only write data that is in USAGE DISPLAY format. If numeric, the sign should be separate, and not overlaid.

Relative

Relative files are declared as ORGANIZATION RELATIVE.

In relative files, each record in the file may be accessed randomly by record number.

Elastic COBOL supports its own format, the Micro Focus relative file format using the "mf:" protocol, and the AcuCOBOL relative file format using the "acu:" protocol.

Indexed

Indexed files are declared as ORGANIZATION INDEXED.

In indexed files, each record in the file may be accessed randomly by key.

An indexed file may consist of one or more files. Generally, either one file is used for both the keys and data, or one file for keys and a separate file for data.

Elastic COBOL supports its own format, the Micro Focus IDXFORMAT"3" file format using the "mf:" protocol and the AcuCOBOL Vision 4 indexed file format using the "acu:" protocol. Remote file access to AcuConnect is supported using the AcuConnect syntax. C-ISAM access is supported on some platforms in conjunction with native ISAM drivers.

XML

ORGANIZATION XML files are always stored in XML format. XML format is a standard, text-based format composed of hierarchical tags and narrative text. The Elastic COBOL extensions for XML allow the reading of XML data.

Reading XML data requires parsing the XML data and extracting its content, so use the ORGANIZATION XML to ease this burden. Writing XML data is currently done using LINE SEQUENTIAL files. Reading XML requires handling unknown types, and this is handled transparently when using ORGANIZATION XML. Writing XML uses only known data from the program so can be handled directly from LINE SEQUENTIAL.

XML Background

XML is a descendent of SGML, like HTML, but more rigorously implemented than HTML without the complexities of SGML. XML is human-readable, with all data stored in text format. XML is machine-readable, with enforceable structure.

XML has nothing but custom tags, tags invented for a particular purpose. For a card catalog, the tags may include title, publisher, and description. For a customer list, the data may include the customer name, the date of first and last purchase, and the average purchase price. Each of these data items is tagged. New information may be added later without breaking the existing format. Some basic validation of structure may be built into the file itself, enforced automatically upon read.

An XML application is not a program, but a set of tags and attributes, rigorously defined. One XML application is XHTML, a more rigorous version

of HTML. Other XML applications exist for a large number of different industries and needs. An XML application may already exist for the industry or need in question, or a new application may be developed by defining the tags and attributes necessary.

Structure of XML

An XML file looks like the following narrative document:

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes">
<biography>
<person scientist="yes">Albert Einstein</person> discovered relativity.
</biography>
```

or the following hierarchical document.

```
<?xml version="1.0" encoding="US-ASCII" standalone="yes">
<customer-database>
        <customer>
                <first-name>Jack</first-name>
                <last-name>Smith</last-name>
                <address-1>123 Elm St.</address-1>
                <city>Springfield</city>
                <state>IL</state>
                <zip>12345</zip>
        </customer>
        <customer>
                <first-name>Jill</first-name>
                <last-name>White</last-name>
                <address-1>234 Maple St.</address-1>
                <city>Springfield</city>
                <state>IL</state>
                <zip>12345</zip>
        </customer>
</customer-database>
```

The first line of the file indicates with a processing instruction, starting with an angle bracket then question mark, that this is an XML file, and the encoding for the file, and whether this file stands alone. This XML processing line is not absolutely required for all files, but it should always be present.

In both cases, the tags indicated by <> characters markup the document with a description of the content. The content is between the tags, such as Albert Einstein or Springfield. An attribute is a name/value pair within a tag, such as scientist = "yes".

Unlike HTML, each tag must have an end tag, beginning with </, or be marked as both a start and end tag by including the / at the end of tag, like <hr/>. Each attribute must have a value, and the value must be in either single- or double-quotes; it cannot be left unquoted. Each document must have a root element, a single tag which surrounds the document data. (In XHTML, for instance, this is <html> surrounding all the HTML.)

Also unlike HTML, each tag and attribute is case-sensitive. A <person> is different from a <Person>.

The white space between tags in an XML file is generally ignored, but some parsers can recognize it.

Parsing XML

Parsing XML is the reading of XML, separating the structure from the contents, determining that the document is well-formed and possibly validating the contents against a DTD, a document type definition. A well-formed document follows all the rules of XML. A valid document follows all the rules of an XML application. (A document not well-formed cannot be valid.) A DTD can help ensure that an invalid document is recognized as such, but it cannot ensure that the data itself is valid. That is, it can ensure that a person has a birth-date child element, but it cannot ensure that the birth-date is not in the future when it's expected to be in the past; that is the job of the program handling the data.

The current generation of parses can handle namespaces. Namespaces are the separation of XML application domains into separate spaces from one another. For example, this allows a single document to have a customer:description and a product:description without conflict. The customer: and product: are namespaces.

XML in Elastic COBOL

XML in Elastic COBOL is defined as a file with ORGANIZATION XML. An XML file is largely like a record sequential file in usage; the only access mode currently supported is SEQUENTIAL.

The tag, such as <person> is the basic structural element driving the remainder of the parsing. The tag has zero or more attributes, zero or more characters of data until the end-tag is reached, and the parser may recognize that it is at a certain location in the file.

The tag structure of the application and the attributes should be known before reading or writing the file. The values of the attributes and contents of the tags is the unknown that the READ can read.

Not all tags used in a document must be referenced from the program code. This allows a future revision of the document to include new tags without breaking the older programs; the new tags will simply be ignored. This allows a customer database to add a field giving the customer birth-date without interfering with the billing program; both a birthday card program and billing can run off the same file.

Any tags used must in the correct hierarchical order. That is, if a person has an attribute of birth-date, the attribute must be a child of person. If a person has character content describing the person, that the description must be a child of person. The hierarchical order is important; filling in pieces irrelevant to the program is not important.

An XML file does not directly have the concept of a record. Elastic COBOL forms XML structures into records on the fly, initializing record items which are not defined within a particular record. Elastic COBOL will form a new record when a second level (level below the document root) is defined. Elastic COBOL will also form a new record when a tag is repeated within a record. Generally, data which a COBOL program would consider handling will be well-formed for record handling.

When reading an XML file, Elastic COBOL tracks the tags in order. If <a><b><c val=1></c><c val=2></c> is found, then two records are found; one is <a><b><c val=1>, the second is <a><b><c val=2>. The higher levels of hierarchy are preserved; when a repetition is found, it is treated as a separate record. By treating the file in this fashion, normal COBOL records are generated; the COBOL code does not need to do separate tracking of the higher-level structural elements.

If records are found in the file that does not match the structure mentioned in the COBOL program, they are skipped. If attributes are found that do not match the structure mentioned in the COBOL program, they are skipped. By only reading in the data which is mentioned, the code is protected from changes in the file or file format.

An XML data record is highly dependent upon its structure, not its byte positions. Certain words are used to mark the structural elements of the XML record. The hierarchical structure of the record is used to map to the hierarchical structure of the XML data.

The name of each data item is important and it must match the XML name exactly; use the IDENTIFIED BY clause to name it exactly. Remember that XML is case sensitive and dashes and underscores are considered separate characters.

Because the naming and record order is considered the important attribute of an XML record, do not redefine or implicitly redefine an XML record (multiple level 01). It is acceptable to make the content-data a group item containing data which is redefined.

The IDENTIFIED BY clause may include IDENTIFIED BY ANY to indicate that any text will match. Which tag was actually found will be returned in the content-name or attribute-name property.

Three forms are available for each of the data items available for an XML tag, a quick form, a short form and a long form. Any may be used, even within a single record; the long provides more information.

An example of the tag format is given below:

Quick Format

```
05 TAG [IDENTIFIED BY "tag-name"] PIC X(n) VALUE content-data.
```

Short format

```
05 TAG [IDENTIFIED BY "tag-name"] [PIC X(n) VALUE].
  10 PCDATA CONTENT PIC X(n) VALUE content-data.
  10 ATTRIBUTE-NAME ATTRIBUTE  [IDENTIFIED BY "attribute-name"]
       PIC X(n) VALUE attribute-data.
  10 LOCATOR-ITEM LOCATOR PIC X(n) VALUE locator-system.
```

Long format

```
05 TAG [IDENTIFIED BY {ANY | "tag-name"}] [PIC X(n) VALUE].
  10 PCDATA CONTENT.
  15 PCDATA-NAME CONTENT-NAME PIC X(n) VALUE content-name.
       15 PCDATA-OFFSET-ITEM CONTENT-OFFSET PIC 9(n)
                VALUE content-offset.
       15 PCDATA-LENGTH-ITEM CONTENT-LENGTH PIC 9(n)
                VALUE content-length.
       15 PCDATA-DATA-ITEM CONTENT-DATA PIC X(n)
                 VALUE content-data.
  10 ATTR-NAME ATTRIBUTE [IDENTIFIED BY {ANY | "attribute-name"}]
       15 ATTR-NAME ATTRIBUTE-NAME PIC X(n)
                 VALUE attribute-name.
       15 ATTR-URI ATTRIBUTE-URI PIC X(n)
                 VALUE attribute-uri.
       15 ATTR-LOCAL ATTRIBUTE-LOCAL PIC X(n)
                 VALUE attribute-local.
       15 ATTR-QNAME ATTRIBUTE-QNAME PIC X(n)
                VALUE attribute-qname.
       15 ATTR-TYPE ATTRIBUTE-TYPE PIC X(8) VALUE attribute-type.
       15 ATTR-DATA ATTRIBUTE-DATA PIC X(n) VALUE attribute-data.
       15 ATTR-LENGTH ATTRIBUTE-LENGTH PIC 9(n)
                 VALUE attribute-length.
10 LOCATOR-GROUP LOCATOR.
       15 LOC-PUBLIC LOCATOR-PUBLIC VALUE locator-public.
       15 LOC-SYSTEM LOCATOR-SYSTEM VALUE locator-system.
       15 LOC-LINE LOCATOR-LINE VALUE locator-line.
       15 LOC-COL LOCATOR-COLUMN VALUE locator-column.
```

In each tag, everything in a record declared at a higher level than CONTENT is content information. Everything in a record declared at a higher level than ATTRIBUTE is attribute information. Everything in a record declared at a higher level than LOCATOR is locator information. Only one (1) LOCATOR per XML record is allowed.

CONTENT

The content in a tag is PCDATA, parsed character data. It is all the data between the start tag and the end tag, the actual content of the tag. It has any escape entities already replaced (such as '>' transformed into the greater than sign '>').

Because PCDATA is the most common information desired, all other information for a tag may be omitted; in such a case, the tag itself holds the data. This is sufficient for many simple XML parsing jobs. The following data record shows a simple XML record for reading names from a customer database.

```
01 CUSTOMER-DATABASE IDENTIFIED BY "customer-database".
   05 CUSTOMER-NAME IDENTIFIED BY "name" PIC X(40).
```

This could read three records from the following XML file:

```
<?xml version="1.0">
<customer-database>
        <name>George Washington</name>
        <name>John Adams</name>
        <name>Thomas Jefferson</name>
</customer-database>
```

When using IDENTIFIED BY ANY for the tag, use CONTENT-NAME to discover the name of the tag. This allows a more compact record description, more flexible in its input; this places more of the input burden on the user program.

Attributes

Attributes are items present within a tag. The attributes give information not within the narrative content. The attribute information is always a simple value within quotes. No structure is implied for the attribute itself; it merely describes the tag and content.

The attribute name must match that given in the text exactly. The attribute name is case sensitive. Because of this, the IDENTIFIED BY clause is used to give the exact name, including upper- or lower-case characters, underscores, hyphens, etc. IDENTIFIED BY ANY is acceptable for an attribute. IDENTIFIED BY "*x" where x is a number starting at one (1) is also acceptable.

The ATTRIBUTE-VALUE is the value of the attribute. The ATTRIBUTE-VALUE-LENGTH is the actual length of the value; the PIC X allocated for ATTRIBUTE-VALUE must be long enough to hold the value. If an attribute is a list of tokens, then the tokens will be concatenated into a single piece of text with each token separated by a single space.

URI, the uniform resource identifier, is the namespace URI. The LOCAL-NAME is the local-name portion. The URI and LOCAL-NAME are only meaningful when namespaces are enabled.

QNAME is the fully qualified XML 1.0 name, the name as used before namespaces were conceived.

TYPE is the type of the attribute, always from one of the following: CDATA, ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES, or NOTATION. CDATA is the default for an undeclared attribute.

When using IDENTIFIED BY ANY for the attribute, use ATTRIBUTE-NAME to discover the name of the attribute. This allows a more compact record description, more flexible in its input; this places more of the input burden on the program. A numbered tag may also be used, when IDENTIFIED BY "*#" where # is an integer value greater than or equal to zero. For example, IDENTIFIED BY "*3" will be the third attribute, whatever its name; its name will be discoverable through the ATTRIBUTE-NAME item.

Locator

The Locator is not guaranteed to be implemented by an XML parser. The Locator provides information as to where the tag or data was located. If not implemented by the XML parser, the data will be set to initialized data, SPACES or ZEROES. This information is generally not important for most uses. It should only be used for optional error reporting. Only one (1) LOCATOR per XML record is allowed.

```
LOCATOR-GROUP LOCATOR.
PUBLIC-ID LOCATOR-PUBLIC VALUE SPACES.
SYSTEM-ID LOCATOR-SYSTEM VALUE SPACES.
LINE-NUMBER LOCATOR-LINE VALUE ZEROES.
COLUMN-NUMBER LOCATOR-COLUMN VALUE ZEROES.
LOCATOR-ITEM LOCATOR PIC X(n) VALUE system-id.
```

ASSIGN TO for XML

The ASSIGN TO for XML is given by the following:

```
ASSIGN TO "xml:[xml-driver:][xml-feature…]xml-url"
```

The ASSIGN TO for an XML file specifies that the file is an XML file, what driver to use for the XML parser, and what features to enable or disable.

The driver name is the class name of an XML parser to use. A system default will be used if driver is not given.

The xml-url portion may be a local filename or a remote http: reference; it may be anything the driver accepts for a reader.

Feature is any of the following:

Xml-Feature	Property Name	Description
xml:	XML designator	optional, implied for XML organization
nons:	Namespaces:	namespaces are disabled
ns:	Namespaces:	namespaces are enabled (def)
nonsp:	Namespace-prefixes:	namespaces with prefix attributes are disabled (def)
nsp:	Namespace-prefixes:	namespaces with prefix attributes are enabled
noval:	Validation:	parser is not validating (no validate messages, def)
val:	Validation:	parser is validating (give validate messages)
nows:	Whitespace	ignore ignorable whitespace (def)
ws:	Whitespace	report ignorable whitespace
noskip:	Skipped-Entity	ignore skipped-entity messages (def)
skip:	Skipped-Entity	give skipped-entity messages
nopc:	Processing-Code	ignore processing-code messages (def)
pc:	Processing-Code	give processing-code messages
noprefix:	start- and end-prefix	ignore prefix messages (def)
prefix:	start- and end-prefix	give prefix messages
elem:	element processing	give element processing messages
driver=name:	XML parser driver	specify SAX2 JAXP XML driver explicitly

XML File Error Information

XML allows up to three file status identifiers to be defined. The first is the standard two-digit file status code. The second and third are optional messages.

Not all file status codes are always returned. Some XML specific file codes are returned only when specifically enabled in the ASSIGN. The default set match what most COBOL programs expect.

General format

FILE STATUS numeric-error alphanumeric-file-status-2 alphanumeric-file-status-3

FILE STATUS	DESCRIPTION	FILE-STATUS-2	FILE-STATUS-3	Notes
00	OK (No Error)			Standard return
05	Open Optional Missing			Standard return, only on OPEN
07	Non-Reel requested, irrelevant for XML			Standard return, reel/unit
10	End Of File (EndDocument)			Standard return, only on READ
14	StartElement	uri	local-name / qname	only if element
15	EndElement	uri	local-name / qname	only if element:
16	StartPrefixMapping	prefix	uri	only if prefix
17	EndPrefixMapping	prefix		only if prefix:
18	ProcessingInstruction	target	data	only if pc
21	SkippedEntity	name		only if skip:
11	Sax-warning	message		
20	Sax-error	message		
20	Parse error	message	locator-message	
30	Sax-fatal-error	message	locator-message	std, error permanent
35	Open Non-Opt Missing	message	message	std, only on OPEN
37	Open Mode N/S	message		std, if cannot set features, load driver, etc.
39	Open Attr Mismatch	message		
41	Already open			std, only on OPEN
42	Already closed			std, only on CLOSE
46	Read already errored			std, only on READ
47	Read not input	message		std, only on READ

XML Deployment

XML parsing in Elastic COBOL requires an external parser. This allows companies concentrating exclusively on XML to produce the best possible XML parsers that are then usable directly from Elastic COBOL. One is

supplied with Elastic COBOL for development, so a development environment requires nothing additional.

However, deployment may need an XML parser included if JAXP (Java API's for XML Processing) is not supported directly by the desired Java Virtual Machine (JVM). JAXP is an optional package for JDK 1.1.8 and higher. JAXP is included in Java 2 Standard Edition 1.4 and higher, and Java 2 Enterprise Edition 1.3 and higher. If not deploying to a JVM that already includes JAXP, be sure to deploy it with the application.

The current version of JAXP is available from Sun at:

http://java.sun.com/xml/xml_jaxp.html

ORGANIZATION XML files also take an additional processing thread. Usually, this is not important for usage, but it can help parsing times dramatically on multi-processor systems. Also, some environments may forbid multiple threads; this would forbid using ORGANIZATION XML files. The XML thread is always a child thread of the COBOL thread doing the OPEN verb.

Transaction

ORGANIZATION TRANSACTION files are for OS/400 DDS compatibility. They reference a terminal display and allow user input/output. Support for these files is dependent upon the DDS Plug-in. Elastic COBOL may attempt to compile ORGANIZATION TRANSACTION files, but the runtime support for them will only be present when the DDS Plug-in is present. See the DDS Plug-in for more information.

Remote File Access

Elastic COBOL supports the use of files from remote servers. The Elastic COBOL remote file server, NFS, AcuConnect and OS/400 remote files are all available.

Remote File Server

The Elastic COBOL remote file server may be started from the IDE for testing. For deployment to a server by itself, it is started as the Java program *com.heirloomcomputing.ecs.exec.fileserver*; it is included within the Elastic COBOL runtime (ecobol.jar) and the runtime must be included for such deployment.

The directory where the remote file server is started determines the current directory for the remote file access. A filename with no directory information will be in the current directory of the file server.

To access files from the remote file server, the Elastic COBOL program should include the protocol 'remote:' in the file assignment. So, the file assigned to "remote:myhost.com:myfile" will use the file named myfile

located on the remote file server myhost.com. The file server must be running at the time to successfully open the file.

Locking

As a standard locking mechanism had not been defined for Java through Java 1.3, Elastic COBOL defines its own locking mechanism. In this mechanism, the two levels of file locking and record locking are handled separately.

File Locking

File locking is done through a renaming scheme. The basic filename, defined in the ASSIGN, represents the file in its inactive, totally unopened state. In order to open a file for the first time, the system must be able to rename it appropriately; after that, only other processes trying to open the file in compatible ways will find the renamed file.

The renamed file always starts with a prefix and underscore before the base filename. The following prefixes are currently used:

Prefix	Description
no_	NO OTHER
aoi_	ALL OTHER (Opened originally for input)
aoo_	ALL OTHER (Opened originally for other than input)
roi_	READ ONLY (Opened originally for input)
roo_	READ ONLY (Opened originally for other than input)

If a process leaves behind an unclosed, locked file, it may be renamed to the base filename to unlock it.

This locking scheme is sufficient to support COBOL 2002 locking conventions. Use only the locking modes absolutely required as future alternative locking systems using native locking conventions may not support all COBOL 2002 modes.

Elastic COBOL's file locking will lock out all other processes attempting to open the base filename unless they explicitly recognize the locking scheme.

Record Locking

Record locking is performed through the use of a record server. The record server must be running at the time the program needs a lock; if not, the program will prompt for the record server to be started.

The record server is located in the Elastic COBOL runtime (ecobol.jar), and is started as the program *com.heirloomcomputing.ecs.exec.RecordServer*. It may be run continuously in the background and it serves all Elastic COBOL programs on the machine.

Elastic COBOL record locks are currently not compatible with other locks on the system, so may only be safely used to lock between Elastic COBOL programs. For locking between languages and machines, SQL is the preferred method.

SQL

Overview

SQL is not another file type, like sequential or indexed. Rather, it is the Structured Query Language, an access method capable of accessing relational databases and other database-like storage.

SQL statements themselves are embedded within the COBOL source code. The SQL statements themselves are not COBOL statements, but rather work with the COBOL code to manipulate the database.

Each of these SQL statements begins with EXEC SQL and ends with END-EXEC. The SQL statement itself is between the EXEC and END-EXEC. The SQL statements may not be streamed together; each must have its own EXEC and END-EXEC.

Some COBOL systems use a pre-compiler to compile the COBOL with SQL code into plain COBOL code, where the COBOL code may have certain calls out to system functions to perform the SQL operations. Rather than using a separate pre-compiler, Elastic COBOL supports embedded SQL directly, compiling it to support JDBC code.

JDBC

As Elastic COBOL compiles the SQL code directly into JDBC, it relies on JDBC drivers for the communication with the database. JDBC is Java Database Connectivity, a standard mechanism by which programs executing in the Java environment may connect to any database that has a JDBC driver. For programmers coming from other environments, JDBC is very similar in purpose to ODBC.

The JDBC driver does the work of converting from the standardized, external form of data access to the proprietary, internal form expected by the database. To access any database from Elastic COBOL, a JDBC driver is required.

This JDBC driver will generally be included with the database or available from the database vendor, but there are also third-party drivers available for certain databases. JDBC drivers are available for all major databases.

The JDBC driver will be a Java executable, meaning it will come in .class, .jar or .zip format. The driver must be in the CLASSPATH to be found and used. The driver may have additional requirements particular to the implementation; any such requirements will be found with the driver documentation.

Just as databases vary in capability, so do the JDBC drivers that access them. Elastic COBOL's SQL access is only as good as the database and JDBC driver used to handle the access. If requiring a particular capability,

be sure that both the SQL database itself and the JDBC driver supports the required action.

SQL Connection

To use SQL with a database, the program must have a connection to the database. Just like files must be opened and closed, the program must connect to and disconnect from the database.

In some COBOL systems, only one SQL database is permissible, so no explicit connection statement is required. However, the flexibility of Elastic COBOL also demands that an explicit connection be made using the SQL CONNECT statement; the database should also be properly disconnected at the end using the SQL DISCONNECT statement.

The format of the CONNECT statement is:

```
EXEC SQL
        CONNECT TO job-url
        [AS connection-name]
        [DRIVER job-driver-name]
        [USER user]
        [PASSWORD|USING password]
END-EXEC
```

A successful connect does require at minimum the jdbc-url, the driver name, and any user name or password required by the database. The information on the jdbc-url formation and driver name is available with the driver documentation.

The jdbc-url is the Internet name of the database; it should be in quotes. In the driver documentation, it will begin with jdbc:, but Elastic COBOL will insert that portion automatically if missing. After the jdbc:, it will have a driver identification protocol name, a colon (:), and any other information required by the driver such as the database name, location, port, etc.

In Java 2, the jdbc-url may also be a Java Data Source name; this is particularly useful with Enterprise JavaBean environments where a connection already exists in the environment, and it must simply be named. To use a data source, use 'ds:name' with name as the named resource.

The jdbc-driver-name is the classname of the JDBC driver. This is the name used to activate the driver. In the JDBC driver documentation, this will be the name in quotes documented as Class.forName("driver"). It will often be a packaged name using the reverse Internet domain, such as 'COM.ibm.db2.jdbc.net.DB2Driver'.

The connection_name is an identifier for later use by other SQL statements; it is optional.

The USER and PASSWORD are the user name and password required for access to the database. These may be omitted for public databases with no user name or password. USING is a synonym for PASSWORD.

To disconnect from the database, use the disconnect command as follows:

```
EXEC SQL
  DISCONNECT
END-EXEC
```

Host Variables

Elastic COBOL supports access to host variables, the COBOL source code variables encapsulating the embedded SQL program. This syntax is the same as that supported by other embedded SQL implementations.

These host variables are accessed using a colon (:) followed by the name of the variable. To check the null indicator for a variable, add another colon (:) and name of the indicator variable.

Host variables should be uniquely defined for best compatibility, but Elastic COBOL supports dot (.) syntax for accessing qualified variable names as well. Where a COBOL program would use 'ALPHA OF BETA' to qualify a name, the SQL host variable would be ':BETA.ALPHA'. The colon signifies the host variable, the beta is the group name, and the alpha is the elementary item within the group.

SQL response codes

The result of a SQL operation may be detected by program code using the SQLCODE or SQLSTATE variables. The SQLCODE has been superceded by SQLSTATE over time, so SQLSTATE should be used in new code wherever possible.

SQLCODE may be defined as a signed numeric of at least three digits. It has only three defined values, indicating success (0), warning (100) and failure (<0).

SQLSTATE may be defined as an alphanumeric of at least five characters. It has the major class of the state in the first two characters with a sub-classification of the state in the next three characters.

Neither SQLCODE nor SQLSTATE is required; they will simply be inaccessible if not defined.

The actual meanings for the values of both SQLCODE and SQLSTATE are defined by the database vendor. Most states are passed directly from the database back to the program. However, there are some standard states defined for SQLSTATE which apply to most databases. These states are listed below.

A SQLCA file is not included with Elastic COBOL to prevent conflicts with existing files, but one may be created for use by EXEC SQL INCLUDE SQLCA END-EXEC. Its preferred contents are:

```
01 SQLCODE PIC S999.
01 SQLSTATE PIC X(5).
```

Chapter 6 – Communication

MQSeries

IBM's Message Queuing system (MQSeries) is available for several platforms operating platforms. Elastic COBOL MQSeries support requires MQSeries from IBM including the MQSeries Java support. Elastic COBOL provides a COBOL interface across all Java platforms by mapping the COBOL interface to the Java interface. Elastic COBOL only supports version 1 MQSeries structures.

Two versions of the MQSeries objects. *com.ibm.mqbind.** objects are only for server programs and *com.ibm.mq.** objects are for client programs. The objects names are the same, but the objects are not interchangeable.

For the purpose of the Elastic COBOL documentation, the default directory is assumed to be used. MQSeries COBOL copy files are located in *x:\mqm\tools\COBOL\copybook*, where x:\ is the MQSeries drive in Windows. MQSeries copy files are located in similar locations for other platforms. Refer to MQSeries documentation for more detailed information concerning location and general operation of MQSeries.

Elastic COBOL support for MQSeries uses parameters according to parameter positions, not according to byte offset and length. It is important to always use the actual MQSeries copy files to pass information.

Application Setup

MQSERVER must be specified as an invocation switch.

Example:

java application_name MQSERVER=true

Local Client Setup

Invocation parameters are Elastic COBOL parameters, included on the command line below.

Example:

java appletname MQ_HOSTNAME=tcp.ip.addr.x
CHANNEL=CHANNEL_name
MQ_PORT=port#

Applet Client Setup

All of the class files used in the applet must be included in subdirectories under the html subdirectory:

ibm.mq.client

classes must be located under html as *com.heirloomcomputing.ecs.exec*

...html/ibm/mq/client

The program must be compiled using the -html option only ONCE to create the html file for invoking the applet.

appletname.html must be edited to add invocation parameters.

Example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Java Applet amq0put1</TITLE>
</HEAD>

<BODY>
<H1>amq0put1</H1>
<APPLET CODE="sqAMQABC.class" CODEBASE="." NAME="amqABC
WIDTH=600 HEIGHT=420 ALIGN=MIDDLE HSPACE=1 VSPACE=1>
<PARAM NAME="WHEN-GENERATED" VALUE="19980818163651000000000">
<PARAM NAME="ECOBOL-MAJOR" VALUE="2">
<PARAM NAME="ECOBOL-MINOR" VALUE="0">
<PARAM NAME="MQ_HOSTNAME" VALUE="tcp.ip.addr.x">
<PARAM NAME="MQ_PORT" VALUE=port#>
<PARAM NAME="MQ_CHANNEL" VALUE="ChannelName">
<PARAM NAME="MQ_USER_ID" VALUE="userid">
<PARAM NAME="MQ_PASSWORD" VALUE="password ">

</APPLET>
<P>
</BODY>
</HTML>
```

The HTML example illustrated was created using:

```
ecobol amqABC.cbl -html –app sqABC
```

MQ_HOSTNAME, MQ_PORT, MQ_CHANNEL, MQ_USER_ID, and MQ_PASSWORD are invocation parameters. The skeleton was then modified to add the appropriate invocation parameters.

MQSeries API's

MQBACK - Back out changes

Data definition:

```
HCONN    PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQBACK' USING HCONN, COMPCODE, REASON.
```

MQBEGIN - Begin unit of work

Data definition:

```
HCONN    PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

MQCLOSE - Close object

Data definition:

```
HCONN    PIC S9(9) BINARY
HOBJ     PIC S9(9) BINARY
OPTIONS  PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQCLOSE' USING HCONN, HOBJ, OPTIONS, COMPCODE, REASON.
```

MQCMIT - Commit changes

Data definition:

```
HCONN    PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQCMIT' USING HCONN, COMPCODE, REASON.
```

MQCONN - Connect queue manager

Data definition:

Required for clients:

```
MQ_HOSTNAME
MQ_PORT
MQ_CHANNEL
MQ_USER_ID
MQ_PASSWORD
```

Required for Applications and Clients:

```
NAME     PIC X(48)
HCONN    PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQCONN' USING NAME, HCONN, COMPCODE, REASON.
```

MQCONNX - Connect queue manager (extended)

Data definition:

```
NAME     PIC X(48)
HCONN    PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQCONNX' USING NAME, HCONN, COMPCODE, REASON.
```

MQDISC - Disconnect queue manager

Data definition:

```
HCONN    PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQDISC' USING HCONN, COMPCODE, REASON.
```

MQGET - Get message

Data definition:

```
HCONN    PIC S9(9) BINARY
HOBJ     PIC S9(9) BINARY
MSGDESC  COPY CMQMDV
BUFFERLENGTH PIC S9(9) BINARY
BUFFER   PIC X(n)
DATALENGTH PIC S9(9) BINARY
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQGET' USING HCONN, HOBJ, MSGDESC,
GETMSGOPTS, BUFFERLENGTH, BUFFERm DATALENGTH, COMPCODE, REASON.
```

MQINQ - Inquire about object attributes

Data definition:

```
HCONN    PIC S9(9) BINARY
HOBJ     PIC S9(9) BINARY
SELECTORCOUNT    PIC S9(9) BINARY
01 SELECTORS-TABLE.
02 SELECTORS PIC S9(9) BINARY OCCURS n TIMES.
INTATTRCOUNT PIC S9(9) BINARY
01 INTATTRS-TABLE
02 INTATTRS PIC S9(9) BINARY OCCURS n TIMES.
CHARATTRLENGTH PIC S9(9) BINARY.
CHARATTRS PIC X(n).
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQINQ' USING HCONN, HOBJ, SELECTORCOUNT,
SELECTORS-TABLE, INTATTRCOUNT,INTATTRS-TABLE,
CHARATTRLENGTH, CHARATTRS, COMPCODE, REASON.
```

MQOPEN - Open object

Data definition:

```
HCONN    PIC S9(9) BINARY
OBJDESC. COPY CMQODV.
OPTIONS PIC S9(9) BINARY.
HOBJ PIC S9(9) BINARY.
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQOPEN' USING HCONN, OBJDESC, OPTIONS, HOBJ, COMPCODE,REASON.
```

MQPUT - Put message

Data definition:

```
HCONN    PIC S9(9) BINARY
HOBJ     PIC S9(9) BINARY
MSGDESC  COPY CMQMDV.
01 PUTMSGOPTS. COPY CMQPMOV.
BUFFERLENGTH PIC S9(9) BINARY
BUFFER   PIC X(n)
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQPUT' USING HCONN, HOBJ, MSGDESC,
PUTMSGOPTS, BUFFERLENGTH, BUFFER, COMPCODE, REASON.
```

MQPUT1 - Put one message

Data definition:

```
HCONN    PIC S9(9) BINARY
OBJDESC. COPY CMQODV.
MSGDESC  COPY CMQMDV.
PUTMSGOPTS. COPY CMQPMOV.
BUFFERLENGTH PIC S9(9) BINARY
BUFFER   PIC X(n)
COMPCODE PIC S9(9) BINARY
REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQPUT1' USING HCONN, OBJDESC, MSGDESC,
PUTMSGOPTS, BUFFERLENGTH, BUFFER, COMPCODE, REASON.
```

MQSET - Set object attributes

Data definition:

```
HCONN    PIC S9(9) BINARY
HOBJ     PIC S9(9) BINARY
SELECTORCOUNT    PIC S9(9) BINARY
 01 SELECTORS-TABLE.
   02 SELECTORS PIC S9(9) BINARY OCCURS n TIMES.
     INTATTRCOUNT PIC S9(9) BINARY
 01 INTATTRS-TABLE
   02 INTATTRS PIC S9(9) BINARY OCCURS n TIMES.
     CHARATTRLENGTH PIC S9(9) BINARY.
     CHARATTRS PIC X(n).
     COMPCODE PIC S9(9) BINARY
     REASON   PIC S9(9) BINARY
```

Call interface:

```
CALL 'MQSET' USING HCONN, HOBJ, SELECTORCOUNT,
SELECTORS-TABLE, INTATTRCOUNT, INTATTRS-TABLE, CHARATTRLENGTH,
CHARATTRS, COMPCODE, REASON.
```

If MQSeries has been installed in the default directory, copy files for MQSeries are located in x:\mqm\tools\COBOL\copybook, where x: is the default drive.

CICS Client

Requirements

Workstation products

Windows operating system.

IBM Communications Server for Windows

CICS Client

Client/Server Programming IBM Document Number SC33-1435-02

CICS Client information refer to the following
http://www.software.ibm.com/ts/cics/platforms/clients/cli204s7.html

Jgate -- Java Gateway for CICS. For information and download a copy of Jgate

http://www.software.ibm.com/ts/cics/platforms/internet/cicsgw4j/announce/jgann201.html

and

http://www.software.ibm.com/ts/cics/platforms/internet/tgw30/ctgann30.html

OS390 products

VTAM

CICS

COBOL

CEE

CICS Overview

CICS/390 is a server that enables CICS clients to invoke CICS transactions and receive the transaction output.

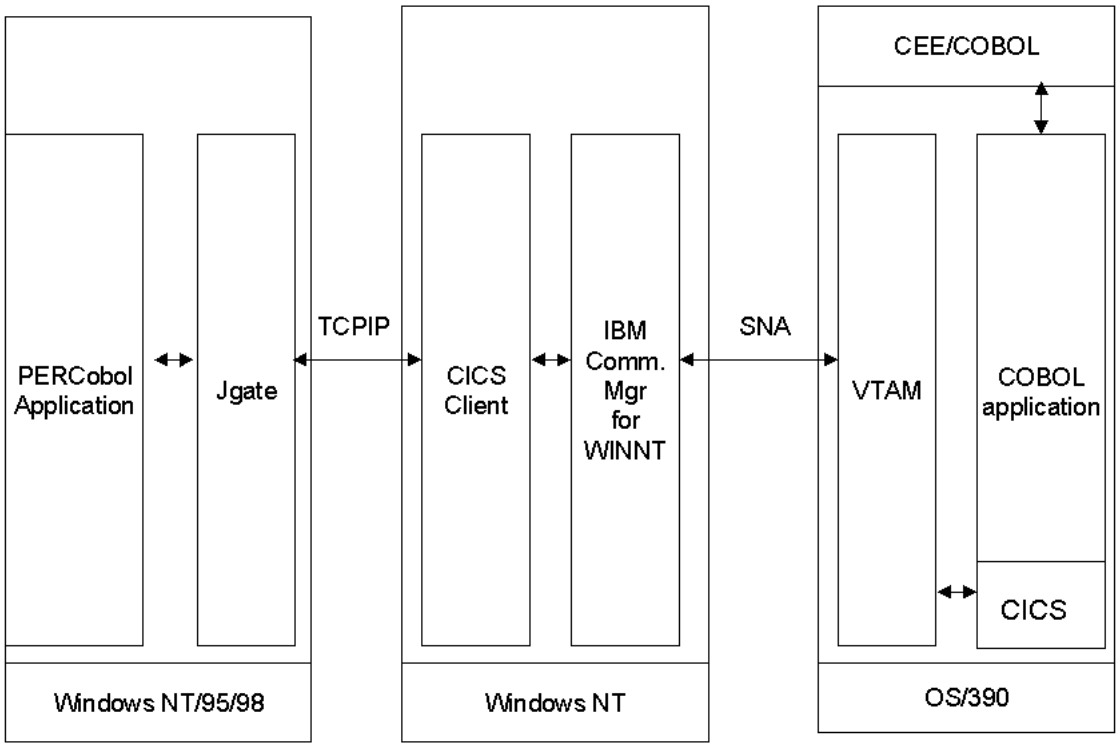


Figure 1. is a high level diagram of CICS Client from Windows to CICS on OS/390.

Left to right, the Client computer has an Elastic COBOL compiled program using CICS ECI call interface to send a CICS request to Jgate. Jgate uses TCPIP to send the request to IBM's CICS Client on Windows Server. CICS Client uses internal interfaces to send the request to IBM Communications Server for Windows. IBM's Communication Server uses SNA/VTAM to send the request to CICS on OS/390. CICS analyzes the request, invokes the CICS transaction and returns the COMMAREA to the client retracing the path to its origin.

The Elastic COBOL application and Jgate can be located on the Windows Server computer or a separate computer as illustrated.

Invocation of COBOL program compiled with Elastic COBOL, javac, and executed using JVM. Location of CICS CLIENT is specified as invocation parameter as CICS_CLIENT=tcp://192.168.0.15:2006/.

The format of CICS_CLIENT data is

protocol://ip_address:port/

protocol - tcp

ip address - address of the CICS Client server

port address - server listening port number

: - paramater separators

/ and // - required delimiters

Note: Imbedded blanks are not allowed.

```
ecobol programname.xxx -app ssprogramename

javac -classpath %CLASSPATH% ssprogramname.java

java  -cp %CLASSPATH%   ssprogramname
CICS_CLIENT=tcp://192.168.0.15:2006/
```

Elastic COBOL CICS Client support uses a ECI call interface to send and receive data from CICS. Descriptions of each call function-type is provided in "CICS Family: Client/Server Programming".

CICS Example

Examples of COBOL programs using CICS ECI interface follow:

```
/*****************************************************************/
/*****************************************************************/
                    CICS Client ECI SYNC
/*****************************************************************/
/*****************************************************************/
     IDENTIFICATION DIVISION.
     PROGRAM-ID.  CICSSYNC.
     ENVIRONMENT DIVISION.
     CONFIGURATION SECTION.
     SOURCE-COMPUTER.  xyz.
     OBJECT-COMPUTER.  xyz.
     DATA DIVISION.
     WORKING-STORAGE SECTION.

     01  SQL-COD PIC S9(9) DISPLAY SIGN LEADING SEPARATE.
     01  UID     PIC X(9) .
     01  HCONN   PIC X(9) .
     01  OBJDESC PIC X(9) .
     01  OPTIONS PIC X(9) .
     01  HOBJ    PIC X(9) .
     01  COMPCODE PIC X(9) .
     01  REASON  PIC X(9) .
     01  D       PIC X(9) .
     01  COUT PIC 9(4) COMP-5 VALUE 0.
     ************************************************************
     *                                          *
     * MODULE NAME        CICSECI.CBL                    *
     *                                          *
     * DESCRIPTIVE NAME   CICS External Call Interface       *
     *                                          *
```

```
* Statement:        Licensed Materials - Property of IBM
*
*                (c) Copyright IBM Corporation 1994,1997
*
*                See Copyright Instructions.
*
*                All rights reserved.
*
*                U.S. Government Users Restricted Rights -
*                use duplication or disclosure restricted
*                by GSA ADP Schedule Contract with
*                IBM Corp.
*
* Status:          Version 2 Release 0
*
* NOTES :-
*
* This copybook is provided with the CICS Client.
*
************************************************************
*
*
* Parameter block for ECI
*

 01 ECI-PARMS.
    05 ECI-CALL-TYPE                  PIC S9(4) COMP-5.
          88 ECI-SYNC-CALL              VALUE 0.
          88 ECI-ASYNC-CALL             VALUE 1.
          88 ECI-SYNC-PARALLEL          VALUE 2.
          88 ECI-ASYNC-PARALLEL         VALUE 3.
          88 ECI-SYNC                 VALUE 516.
          88 ECI-ASYNC                VALUE 517.
          88 ECI-ASYNC-NOTIFY-MSG       VALUE 518.
          88 ECI-ASYNC-NOTIFY-SEM       VALUE 519.
          88 ECI-GET-REPLY            VALUE 520.
          88 ECI-GET-REPLY-WAIT         VALUE 521.
          88 ECI-STATE-SYNC             VALUE 522.
          88 ECI-STATE-ASYNC            VALUE 523.
          88 ECI-STATE-ASYNC-SEM        VALUE 524.
          88 ECI-STATE-ASYNC-MSG        VALUE 525.
          88 ECI-GET-SPECIFIC-REPLY     VALUE 528.
          88 ECI-GET-SPECIFIC-REPLY-WAIT  VALUE 529.
    05 ECI-PROGRAM-NAME               PIC X(8).
    05 ECI-USERID                   PIC X(8).
    05 ECI-PASSWORD                   PIC X(8).
    05 ECI-TRANSID                  PIC X(4).
    05 ECI-ABEND-CODE                 PIC X(4).
    05 ECI-COMMAREA                   POINTER.
    05 ECI-COMMAREA-LENGTH              PIC S9(4) COMP-5.
    05 ECI-TIMEOUT                  PIC S9(4) COMP-5.
    05 ECI-SYS-RETURN-CODE              PIC S9(4) COMP-5.
    05 ECI-EXTEND-MODE                PIC S9(4) COMP-5.
          88 ECI-NO-EXTEND            VALUE 0.
          88 ECI-EXTENDED             VALUE 1.
          88 ECI-CANCEL               VALUE 2.
          88 ECI-COMMIT               VALUE 2.
          88 ECI-BACKOUT              VALUE 3.
          88 ECI-STATE-IMMEDIATE        VALUE 4.
          88 ECI-STATE-CHANGED          VALUE 5.
          88 ECI-STATE-CANCEL           VALUE 6.
    05 ECI-WINDOW-HANDLE              PIC S9(8) COMP-5.
    05 ECI-SEM-HANDLE REDEFINES ECI-WINDOW-HANDLE
```

```
                                PIC S9(8) COMP-5.
    05 FILLER        REDEFINES ECI-WINDOW-HANDLE.
       10 ECI-MS-WINDOW-HANDLE          PIC S9(4) COMP-5.
       10 ECI-MS-INSTANCE-HANDLE        PIC S9(4) COMP-5.
    05 ECI-MESSAGE-ID                 PIC  9(4) COMP-5.
    05 ECI-MESSAGE-QUALIFIER          PIC S9(4) COMP-5.
    05 ECI-LUW-TOKEN                  PIC S9(8) COMP-5.
          88 ECI-LUW-NEW              VALUE 0.
    05 ECI-SYSID                   PIC X(4).
    05 ECI-VERSION                 PIC S9(4) COMP-5.
          88 ECI-VERSION-0            VALUE 0.
          88 ECI-VERSION-1            VALUE 1.
          88 ECI-VERSION-1A           VALUE 2.
          88 ECI-VERSION-MAX          VALUE 2.
    05 ECI-SYSTEM-NAME                PIC X(8).
    05 ECI-CALLBACK            PROCEDURE-POINTER.
    05 ECI-USERID2                  PIC X(16).
    05 ECI-PASSWORD2                PIC X(16).
    05 ECI-TPN                   PIC X(4).
*    05 ECI-COMMAREA POINTER.
*
* List of error returns from CICSEXTERNALCALL
*

 01 ECI-ERROR-ID                   PIC S9(4) COMP-5.
    88 ECI-NO-ERROR                VALUE  0.
    88 ECI-ERR-INVALID-DATA-LENGTH      VALUE -1.
    88 ECI-ERR-INVALID-EXTEND-MODE      VALUE -2.
    88 ECI-ERR-NO-CICS              VALUE -3.
    88 ECI-ERR-CICS-DIED            VALUE -4.
    88 ECI-ERR-REQUEST-TIMEOUT          VALUE -5.
    88 ECI-ERR-NO-REPLY             VALUE -5.
    88 ECI-ERR-RESPONSE-TIMEOUT         VALUE -6.
    88 ECI-ERR-TRANSACTION-ABEND        VALUE -7.
    88 ECI-ERR-EXEC-NOT-RESIDENT        VALUE -8.
    88 ECI-ERR-LUW-TOKEN            VALUE -8.
    88 ECI-ERR-SYSTEM-ERROR             VALUE -9.
    88 ECI-ERR-NULL-WIN-HANDLE          VALUE -10.
    88 ECI-ERR-NULL-MESSAGE-ID          VALUE -12.
    88 ECI-ERR-THREAD-CREATE-ERROR      VALUE -13.
    88 ECI-ERR-INVALID-CALL-TYPE        VALUE -14.
    88 ECI-ERR-ALREADY-ACTIVE           VALUE -15.
    88 ECI-ERR-RESOURCE-SHORTAGE        VALUE -16.
    88 ECI-ERR-NO-SESSIONS          VALUE -17.
    88 ECI-ERR-NULL-SEM-HANDLE          VALUE -18.
    88 ECI-ERR-INVALID-DATA-AREA        VALUE -19.
    88 ECI-ERR-INVALID-VERSION          VALUE -21.
    88 ECI-ERR-UNKNOWN-SERVER           VALUE -22.
    88 ECI-ERR-CALL-FROM-CALLBACK       VALUE -23.
    88 ECI-ERR-INVALID-TRANSID          VALUE -24.
    88 ECI-ERR-MORE-SYSTEMS             VALUE -25.
    88 ECI-ERR-NO-SYSTEMS           VALUE -26.
    88 ECI-ERR-SECURITY-ERROR           VALUE -27.
    88 ECI-ERR-MAX-SYSTEMS          VALUE -28.

    88 ECI-ERR-MAX-SESSIONS             VALUE -29.
    88 ECI-ERR-ROLLEDBACK           VALUE -30.

*
* Commarea layout for ECI-STATE-xxx CallType requests other
* than when the  ExtendMode is ECI-STATE-CANCEL.
*
* It should be supplied with valid values for a request where
```

```
* the ExtendMode is ECI-STATE-CHANGED.  In this case a
* response will be returned only when the status is different
*to that which was supplied.
*
* It will be returned with the current status in these fields
* except where  the ExtendMode is ECI-STATE-CANCEL.
*

 01 ECI-STATUS.
    05 ECI-CONNECTION-TYPE              PIC S9(4) COMP-5.
          88 ECI-CONNECTED-NOWHERE        VALUE 0.
          88 ECI-CONNECTED-TO-SERVER      VALUE 1.
          88 ECI-CONNECTED-TO-CLIENT      VALUE 2.
    05 ECI-CICS-SERVER-STATUS            PIC S9(4) COMP-5.
          88 ECI-SERVERSTATE-UNKNOWN      VALUE 0.
          88 ECI-SERVERSTATE-UP           VALUE 1.
          88 ECI-SERVERSTATE-DOWN         VALUE 2.
    05 ECI-CICS-CLIENT-STATUS            PIC S9(4) COMP-5.
          88 ECI-CLIENTSTATE-UNKNOWN      VALUE 0.
          88 ECI-CLIENTSTATE-UP           VALUE 1.
          88 ECI-CLIENTSTATE-INAPPLICABLE  VALUE 2.
    05 FILLER                           PIC S9(4) COMP-5.
    05 FILLER                           PIC S9(4) COMP-5.
*  01 COMMAREA PIC X(300).
*
*  CICSECILISTSYSTEMS.
*
*  Note: The value '16' assigned to CICS-ECINUMSYS and the
*        matching value in the OCCURS clause for
*        CICS-ECISYSTEM may need be increased if the
*        ECI-ERR-MORE-SYSTEMS error occurs.
*

 77 CICS-ECI-SYSTEM-MAX          PIC 9(4) COMP-5 VALUE 8.
 77 CICS-ECI-DESCRIPTION-MAX     PIC 9(4) COMP-5 VALUE 60.

 77 CICS-ECINUMSYS               PIC 9(4) COMP-5 VALUE 16.

 01 CICS-ECISYSTEM.
    02 FILLER
    OCCURS 0 TO 16 TIMES DEPENDING ON CICS-ECINUMSYS.
       05 SYSTEMNAME           PIC X(8).
       05 FILLER               PIC X.
       05 SYSTEMDESC           PIC X(60).
       05 FILLER               PIC X.


 01 COMMAREA2 PIC X(500).
 LINKAGE SECTION.
 01 COMMAREA.
   05 SEND-DATA PIC X(150).
   05 REC-DATA PIC X(150).
 PROCEDURE DIVISION USING COMMAREA.
*PROCEDURE DIVISION.
 P0.

      MOVE "This is the commarea for cics." TO COMMAREA.
*     MOVE "This is the commarea for cics." TO COMMAREA2.

*1      MOVE "DFHZCN1" TO ECI-PROGRAM-NAME.
*1      MOVE "CCIN"  TO ECI-TRANSID.
*1      MOVE "CPMI" TO ECI-TPN.
```

```
*2      MOVE "DFHZCN1" TO ECI-PROGRAM-NAME.
*2      MOVE "CCIN"  TO ECI-TRANSID.
*2      MOVE "CPMI" TO ECI-TPN.

       MOVE "LINKPROG" TO ECI-PROGRAM-NAME.
       MOVE "CPMI"  TO ECI-TRANSID.
*       MOVE "CPMI" TO ECI-TPN.

       MOVE "P390" TO  ECI-USERID.
       MOVE "IBM7TED" TO ECI-PASSWORD.
*
       MOVE "aaaa"    TO ECI-ABEND-CODE.
       SET ECI-COMMAREA TO ADDRESS OF COMMAREA
*      SET  ADDRESS OF COMMAREA TO ECI-COMMAREA.
       MOVE  LENGTH  OF COMMAREA TO ECI-COMMAREA-LENGTH.
       DISPLAY "COBOL.COMMAREA.LENGTH=" ECI-COMMAREA-LENGTH.
       MOVE 00    TO ECI-TIMEOUT.
       MOVE ZERO    TO ECI-SYS-RETURN-CODE.
       MOVE ZERO   TO ECI-EXTEND-MODE.
       MOVE ZERO   TO ECI-WINDOW-HANDLE.
       MOVE ZERO   TO ECI-SEM-HANDLE.
       MOVE ZERO   TO ECI-MESSAGE-ID.
       MOVE ZERO   TO ECI-MESSAGE-QUALIFIER.
       MOVE ZERO   TO ECI-LUW-TOKEN.
       MOVE "sys1" TO ECI-SYSID.
       MOVE 1   TO ECI-VERSION.
       MOVE "P390" TO ECI-SYSTEM-NAME.
       MOVE 1      TO CICS-ECI-SYSTEM-MAX.
*      SET   ECI-CALLBACK TO  ENTRY P0.
       SET   ECI-CALLBACK TO  P1.
*      SET ECI-CALLBACK TO ENTRY P1.
       MOVE "P390SNA " TO ECI-SYSTEM-NAME.
       MOVE "user2 " TO ECI-USERID2.
       MOVE "ps2 " TO ECI-PASSWORD2.

       MOVE "This is the commarea for cics ECI-SYNC."
            TO SEND-DATA.
       SET ECI-SYNC                  TO TRUE.
       SET ECI-NO-EXTEND             TO TRUE.
*      DISPLAY " addr of commarea= " ECI-COMMAREA.
*      DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
       DISPLAY
             " commarea before call to P390\CICS= " COMMAREA.
       CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
       DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
       DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
       DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
       DISPLAY "Extend-Mode=ECI-NO-EXTEND".

       MOVE " " TO REC-DATA.
       MOVE "This is the commarea for cics ECI-SYNC."
            TO SEND-DATA.
       SET ECI-SYNC                  TO TRUE.
       SET ECI-EXTENDED              TO TRUE.
*      DISPLAY " addr of commarea= " ECI-COMMAREA.
*      DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
       DISPLAY
             " commarea before call to P390\CICS= " COMMAREA.
       CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
       DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
       DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
       DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
```

```
           DISPLAY "Extend-Mode=ECI-EXTENDED".

           MOVE " " TO REC-DATA.
           MOVE "This is the commarea for cics ECI-SYNC."
                TO SEND-DATA.
           SET ECI-SYNC                    TO TRUE.
           SET ECI-COMMIT                  TO TRUE.
       *    DISPLAY " addr of commarea= " ECI-COMMAREA.
       *    DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
           DISPLAY
                " commarea before call to P390\CICS= " COMMAREA.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
           DISPLAY "Extend-Mode=ECI-COMMIT".

           MOVE " " TO REC-DATA.
           MOVE "This is the commarea for cics ECI-SYNC."
                TO SEND-DATA.
           SET ECI-SYNC                    TO TRUE.
           SET ECI-BACKOUT                 TO TRUE.
       *    DISPLAY " addr of commarea= " ECI-COMMAREA.
       *    DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
           DISPLAY
                " commarea before call to P390\CICS= " COMMAREA.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
           DISPLAY "Extend-Mode=ECI-BACKOUT".

          STOP RUN.
       P1.
           MOVE "ABC" TO D.
           DISPLAY " P1 entered ".
           DISPLAY " P1 COMMAREA= " COMMAREA.
           DISPLAY " P1 entered ".
  /*****************************************************************/
   /*****************************************************************/
                       CICS Client ECI ASYNC
   /*****************************************************************/
   /*****************************************************************/
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  CICSASYNC.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  xyz.
       OBJECT-COMPUTER.  xyz.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  SQL-COD PIC S9(9) DISPLAY SIGN LEADING SEPARATE.
       01  UID     PIC X(9) .
       01  HCONN   PIC X(9) .
       01  OBJDESC PIC X(9) .
       01  OPTIONS PIC X(9) .
       01  HOBJ    PIC X(9) .
       01  COMPCODE PIC X(9) .
       01  REASON  PIC X(9) .
       01  D       PIC X(9) .
       01  COUT PIC 9(4) COMP-5 VALUE 0.
      ************************************************************
```

```
*                                                               *
* MODULE NAME        CICSECI.CBL
*
* DESCRIPTIVE NAME   CICS External Call Interface
*
* Statement:         Licensed Materials - Property of IBM
*
*                    63H9790
*                    (c) Copyright IBM Corporation 1994,1997
*
*                    See Copyright Instructions.
*
*                    All rights reserved.
*
*                    U.S. Government Users Restricted Rights -
*                    use duplication or disclosure restricted
*                    by GSA ADP Schedule Contract with IBM
*                    Corp.
*
* Status:            Version 2 Release 0
* NOTES :-
*
* This copybook is provided with the CICS Client.
****************************************************************
*
* Parameter block for ECI
*
 01 ECI-PARMS.
    05 ECI-CALL-TYPE                  PIC S9(4) COMP-5.
          88 ECI-SYNC-CALL              VALUE 0.
          88 ECI-ASYNC-CALL             VALUE 1.
          88 ECI-SYNC-PARALLEL          VALUE 2.
          88 ECI-ASYNC-PARALLEL         VALUE 3.
          88 ECI-SYNC                 VALUE 516.
          88 ECI-ASYNC                VALUE 517.
          88 ECI-ASYNC-NOTIFY-MSG       VALUE 518.
          88 ECI-ASYNC-NOTIFY-SEM       VALUE 519.
          88 ECI-GET-REPLY            VALUE 520.
          88 ECI-GET-REPLY-WAIT         VALUE 521.
          88 ECI-STATE-SYNC           VALUE 522.
          88 ECI-STATE-ASYNC          VALUE 523.
          88 ECI-STATE-ASYNC-SEM        VALUE 524.
          88 ECI-STATE-ASYNC-MSG        VALUE 525.
          88 ECI-GET-SPECIFIC-REPLY     VALUE 528.
          88 ECI-GET-SPECIFIC-REPLY-WAIT  VALUE 529.
    05 ECI-PROGRAM-NAME               PIC X(8).
    05 ECI-USERID                   PIC X(8).
    05 ECI-PASSWORD                   PIC X(8).
    05 ECI-TRANSID                  PIC X(4).
    05 ECI-ABEND-CODE                 PIC X(4).
    05 ECI-COMMAREA                   POINTER.
    05 ECI-COMMAREA-LENGTH              PIC S9(4) COMP-5.
    05 ECI-TIMEOUT                  PIC S9(4) COMP-5.
    05 ECI-SYS-RETURN-CODE            PIC S9(4) COMP-5.
    05 ECI-EXTEND-MODE              PIC S9(4) COMP-5.
          88 ECI-NO-EXTEND            VALUE 0.
          88 ECI-EXTENDED             VALUE 1.
          88 ECI-CANCEL               VALUE 2.
          88 ECI-COMMIT               VALUE 2.
          88 ECI-BACKOUT              VALUE 3.
          88 ECI-STATE-IMMEDIATE        VALUE 4.
          88 ECI-STATE-CHANGED          VALUE 5.
          88 ECI-STATE-CANCEL         VALUE 6.
```

```
    05 ECI-WINDOW-HANDLE                PIC S9(8) COMP-5.
    05 ECI-SEM-HANDLE REDEFINES ECI-WINDOW-HANDLE
                                 PIC S9(8) COMP-5.
    05 FILLER        REDEFINES ECI-WINDOW-HANDLE.
       10 ECI-MS-WINDOW-HANDLE          PIC S9(4) COMP-5.
       10 ECI-MS-INSTANCE-HANDLE        PIC S9(4) COMP-5.
    05 ECI-MESSAGE-ID                  PIC  9(4) COMP-5.
    05 ECI-MESSAGE-QUALIFIER           PIC S9(4) COMP-5.
    05 ECI-LUW-TOKEN                   PIC S9(8) COMP-5.
          88 ECI-LUW-NEW               VALUE 0.
    05 ECI-SYSID                     PIC X(4).
    05 ECI-VERSION                   PIC S9(4) COMP-5.
          88 ECI-VERSION-0             VALUE 0.
          88 ECI-VERSION-1             VALUE 1.
          88 ECI-VERSION-1A            VALUE 2.
          88 ECI-VERSION-MAX           VALUE 2.
    05 ECI-SYSTEM-NAME                 PIC X(8).
    05 ECI-CALLBACK             PROCEDURE-POINTER.
    05 ECI-USERID2                   PIC X(16).
    05 ECI-PASSWORD2                 PIC X(16).
    05 ECI-TPN                       PIC X(4).
*    05 ECI-COMMAREA POINTER.
*
* List of error returns from CICSEXTERNALCALL
*

 01 ECI-ERROR-ID                     PIC S9(4) COMP-5.
    88 ECI-NO-ERROR                  VALUE  0.
    88 ECI-ERR-INVALID-DATA-LENGTH       VALUE -1.
    88 ECI-ERR-INVALID-EXTEND-MODE       VALUE -2.
    88 ECI-ERR-NO-CICS                VALUE -3.
    88 ECI-ERR-CICS-DIED              VALUE -4.
    88 ECI-ERR-REQUEST-TIMEOUT           VALUE -5.
    88 ECI-ERR-NO-REPLY               VALUE -5.
    88 ECI-ERR-RESPONSE-TIMEOUT          VALUE -6.
    88 ECI-ERR-TRANSACTION-ABEND         VALUE -7.
    88 ECI-ERR-EXEC-NOT-RESIDENT         VALUE -8.
    88 ECI-ERR-LUW-TOKEN              VALUE -8.
    88 ECI-ERR-SYSTEM-ERROR              VALUE -9.
    88 ECI-ERR-NULL-WIN-HANDLE           VALUE -10.
    88 ECI-ERR-NULL-MESSAGE-ID           VALUE -12.
    88 ECI-ERR-THREAD-CREATE-ERROR       VALUE -13.
    88 ECI-ERR-INVALID-CALL-TYPE         VALUE -14.
    88 ECI-ERR-ALREADY-ACTIVE            VALUE -15.
    88 ECI-ERR-RESOURCE-SHORTAGE         VALUE -16.
    88 ECI-ERR-NO-SESSIONS            VALUE -17.
    88 ECI-ERR-NULL-SEM-HANDLE           VALUE -18.
    88 ECI-ERR-INVALID-DATA-AREA         VALUE -19.
    88 ECI-ERR-INVALID-VERSION           VALUE -21.
    88 ECI-ERR-UNKNOWN-SERVER            VALUE -22.
    88 ECI-ERR-CALL-FROM-CALLBACK        VALUE -23.
    88 ECI-ERR-INVALID-TRANSID           VALUE -24.
    88 ECI-ERR-MORE-SYSTEMS              VALUE -25.
    88 ECI-ERR-NO-SYSTEMS             VALUE -26.
    88 ECI-ERR-SECURITY-ERROR            VALUE -27.
    88 ECI-ERR-MAX-SYSTEMS            VALUE -28.
    88 ECI-ERR-MAX-SESSIONS           VALUE -29.
    88 ECI-ERR-ROLLEDBACK             VALUE -30.

*
* Commarea layout for ECI-STATE-xxx CallType requests other
* than when the  ExtendMode is ECI-STATE-CANCEL.
*
```

```
* It should be supplied with valid values for a request where
* the ExtendMode is ECI-STATE-CHANGED.  In this case a
* response will be returned only when the status is different
* to that which was supplied.
*
* It will be returned with the current status in these fields
* except where  the ExtendMode is ECI-STATE-CANCEL.

*

 01 ECI-STATUS.
    05 ECI-CONNECTION-TYPE              PIC S9(4) COMP-5.
          88 ECI-CONNECTED-NOWHERE        VALUE 0.
          88 ECI-CONNECTED-TO-SERVER      VALUE 1.
          88 ECI-CONNECTED-TO-CLIENT      VALUE 2.
    05 ECI-CICS-SERVER-STATUS            PIC S9(4) COMP-5.
          88 ECI-SERVERSTATE-UNKNOWN      VALUE 0.
          88 ECI-SERVERSTATE-UP           VALUE 1.
          88 ECI-SERVERSTATE-DOWN         VALUE 2.
    05 ECI-CICS-CLIENT-STATUS            PIC S9(4) COMP-5.
          88 ECI-CLIENTSTATE-UNKNOWN      VALUE 0.
          88 ECI-CLIENTSTATE-UP           VALUE 1.
          88 ECI-CLIENTSTATE-INAPPLICABLE  VALUE 2.
    05 FILLER                           PIC S9(4) COMP-5.
    05 FILLER                           PIC S9(4) COMP-5.
*  01 COMMAREA PIC X(300).
*
*  CICSECILISTSYSTEMS.
*
*  Note: The value '16' assigned to CICS-ECINUMSYS and the
*  matching value in the OCCURS clause for CICS-ECISYSTEM may
*  need be increased if the ECI-ERR-MORE-SYSTEMS error
*  occurs.
*

 77 CICS-ECI-SYSTEM-MAX          PIC 9(4) COMP-5 VALUE 8.
 77 CICS-ECI-DESCRIPTION-MAX     PIC 9(4) COMP-5 VALUE 60.

 77 CICS-ECINUMSYS               PIC 9(4) COMP-5 VALUE 16.

 01 CICS-ECISYSTEM.
    02 FILLER
    OCCURS 0 TO 16 TIMES DEPENDING ON CICS-ECINUMSYS.
       05 SYSTEMNAME          PIC X(8).
       05 FILLER              PIC X.
       05 SYSTEMDESC          PIC X(60).
       05 FILLER              PIC X.


 01 COMMAREA2 PIC X(500).
 LINKAGE SECTION.
 01 COMMAREA.
   05 SEND-DATA PIC X(150).
   05 REC-DATA PIC X(150).
 PROCEDURE DIVISION USING COMMAREA.
*PROCEDURE DIVISION.
 P0.

      MOVE "This is the commarea for cics." TO COMMAREA.
*     MOVE "This is the commarea for cics." TO COMMAREA2.

*1      MOVE "DFHZCN1" TO ECI-PROGRAM-NAME.
*1      MOVE "CCIN"  TO ECI-TRANSID.
```

```
*1      MOVE "CPMI" TO ECI-TPN.

*2      MOVE "DFHZCN1" TO ECI-PROGRAM-NAME.
*2      MOVE "CCIN"  TO ECI-TRANSID.
*2      MOVE "CPMI" TO ECI-TPN.

        MOVE "LINKPROG" TO ECI-PROGRAM-NAME.
        MOVE "CPMI"  TO ECI-TRANSID.
*       MOVE "CPMI" TO ECI-TPN.

        MOVE "P390" TO  ECI-USERID.
        MOVE "IBM7TED" TO ECI-PASSWORD.
*
        MOVE "aaaa"    TO ECI-ABEND-CODE.
        SET ECI-COMMAREA TO ADDRESS OF COMMAREA
*       SET  ADDRESS OF COMMAREA TO ECI-COMMAREA.
        MOVE  LENGTH  OF COMMAREA TO ECI-COMMAREA-LENGTH.
        DISPLAY "COBOL.COMMAREA.LENGTH=" ECI-COMMAREA-LENGTH.
        MOVE 00    TO ECI-TIMEOUT.
        MOVE ZERO    TO ECI-SYS-RETURN-CODE.
        MOVE ZERO   TO ECI-EXTEND-MODE.
        MOVE ZERO   TO ECI-WINDOW-HANDLE.
        MOVE ZERO   TO ECI-SEM-HANDLE.
        MOVE ZERO   TO ECI-MESSAGE-ID.
        MOVE ZERO   TO ECI-MESSAGE-QUALIFIER.
        MOVE ZERO   TO ECI-LUW-TOKEN.
        MOVE "sys1" TO ECI-SYSID.
        MOVE 1   TO ECI-VERSION.
        MOVE "P390" TO ECI-SYSTEM-NAME.
        MOVE 1      TO CICS-ECI-SYSTEM-MAX.
*       SET   ECI-CALLBACK TO  ENTRY P0.
*       SET   ECI-CALLBACK TO  P1.
        SET ECI-CALLBACK TO NULL.
*       SET ECI-CALLBACK TO ENTRY P1.
        MOVE "P390SNA " TO ECI-SYSTEM-NAME.
        MOVE "user2 " TO ECI-USERID2.
        MOVE "ps2 " TO ECI-PASSWORD2.

        MOVE "This is the commarea for cics ECI-ASYNC."
             TO SEND-DATA.
        MOVE " " TO REC-DATA.
        SET ECI-ASYNC                   TO TRUE.
        SET ECI-NO-EXTEND               TO TRUE.
        DISPLAY
             " commarea before call to P390\CICS= " COMMAREA.
        CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
        DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
        DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
        DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

        MOVE " " TO REC-DATA.
        MOVE "This is the commarea for cics ECI-SYNC."
             TO SEND-DATA.
        SET ECI-ASYNC                   TO TRUE.
        SET ECI-EXTENDED                TO TRUE.
*       DISPLAY " addr of commarea= " ECI-COMMAREA.
*       DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
        DISPLAY
              " commarea before call to P390\CICS= " COMMAREA.
        CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
        DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
        DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
        DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
```

```
           DISPLAY "Extend-Mode=ECI-EXTENDED".

           MOVE " " TO REC-DATA.
           MOVE "This is the commarea for cics ECI-SYNC."
                TO SEND-DATA.
           SET ECI-ASYNC                  TO TRUE.
           SET ECI-COMMIT                 TO TRUE.
       *    DISPLAY " addr of commarea= " ECI-COMMAREA.
       *    DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
           DISPLAY
                " commarea before call to P390\CICS= " COMMAREA.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
           DISPLAY "Extend-Mode=ECI-COMMIT".

           MOVE " " TO REC-DATA.
           MOVE "This is the commarea for cics ECI-SYNC."
                TO SEND-DATA.
           SET ECI-ASYNC                  TO TRUE.
           SET ECI-BACKOUT                TO TRUE.
       *    DISPLAY " addr of commarea= " ECI-COMMAREA.
       *    DISPLAY " address2 of commarea= " ADDRESS OF COMMAREA.
           DISPLAY
                " commarea before call to P390\CICS= " COMMAREA.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.
           DISPLAY "Extend-Mode=ECI-BACKOUT".

          STOP RUN.
       P1.
           MOVE "ABC" TO D.
           DISPLAY " P1 entered ".
           DISPLAY " Commarea may not contain CICS response,".
           DISPLAY " ASYNC doesn't wait for response." COMMAREA.
/*****************************************************************/
 /*****************************************************************/
                    CICS Client ECI Get Specific Reply
 /*****************************************************************/
 /*****************************************************************/
       IDENTIFICATION DIVISION.
       PROGRAM-ID.  CICSGETSPECIFICREPLY.
       ENVIRONMENT DIVISION.
       CONFIGURATION SECTION.
       SOURCE-COMPUTER.  xyz.
       OBJECT-COMPUTER.  xyz.
       DATA DIVISION.
       WORKING-STORAGE SECTION.

       01  SQL-COD PIC S9(9) DISPLAY SIGN LEADING SEPARATE.
       01  UID     PIC X(9) .
       01  HCONN   PIC X(9) .
       01  OBJDESC PIC X(9) .
       01  OPTIONS PIC X(9) .
       01  HOBJ    PIC X(9) .
       01  COMPCODE PIC X(9) .
       01  REASON  PIC X(9) .
       01  D       PIC X(9) .
       01  COUT PIC 9(4) COMP-5 VALUE 0.
      ************************************************************
```

```
    * MODULE NAME      CICSECI.CBL
    * DESCRIPTIVE NAME CICS External Call Interface
    * Statement:       Licensed Materials - Property of IBM
    *
    *               63H9790
    *               (c) Copyright IBM Corporation 1994,1997
    *
    *               See Copyright Instructions.
    *
    *               All rights reserved.
    *
    *               U.S. Government Users Restricted Rights -
    *               use duplication or disclosure restricted by
    *               GSA ADP Schedule Contract with IBM Corp.
    *
    * Status:         Version 2 Release 0
    *
    * NOTES :-
*
* This copybook is provided with the CICS Client.
*
*
************************************************************
*
* Parameter block for ECI
*
 01 ECI-PARMS.
    05 ECI-CALL-TYPE                  PIC S9(4) COMP-5.
          88 ECI-SYNC-CALL              VALUE 0.
          88 ECI-ASYNC-CALL             VALUE 1.
          88 ECI-SYNC-PARALLEL          VALUE 2.
          88 ECI-ASYNC-PARALLEL         VALUE 3.
          88 ECI-SYNC                 VALUE 516.
          88 ECI-ASYNC                VALUE 517.
          88 ECI-ASYNC-NOTIFY-MSG       VALUE 518.
          88 ECI-ASYNC-NOTIFY-SEM       VALUE 519.
          88 ECI-GET-REPLY              VALUE 520.
          88 ECI-GET-REPLY-WAIT         VALUE 521.
          88 ECI-STATE-SYNC             VALUE 522.
          88 ECI-STATE-ASYNC            VALUE 523.
          88 ECI-STATE-ASYNC-SEM        VALUE 524.
          88 ECI-STATE-ASYNC-MSG        VALUE 525.
          88 ECI-GET-SPECIFIC-REPLY     VALUE 528.
          88 ECI-GET-SPECIFIC-REPLY-WAIT  VALUE 529.
    05 ECI-PROGRAM-NAME               PIC X(8).
    05 ECI-USERID                   PIC X(8).
    05 ECI-PASSWORD                   PIC X(8).
    05 ECI-TRANSID                  PIC X(4).
    05 ECI-ABEND-CODE                 PIC X(4).
    05 ECI-COMMAREA                   POINTER.
    05 ECI-COMMAREA-LENGTH              PIC S9(4) COMP-5.
    05 ECI-TIMEOUT                  PIC S9(4) COMP-5.
    05 ECI-SYS-RETURN-CODE              PIC S9(4) COMP-5.
    05 ECI-EXTEND-MODE                PIC S9(4) COMP-5.
          88 ECI-NO-EXTEND              VALUE 0.
          88 ECI-EXTENDED               VALUE 1.
          88 ECI-CANCEL               VALUE 2.
          88 ECI-COMMIT               VALUE 2.
          88 ECI-BACKOUT                VALUE 3.
          88 ECI-STATE-IMMEDIATE        VALUE 4.
          88 ECI-STATE-CHANGED          VALUE 5.
          88 ECI-STATE-CANCEL           VALUE 6.
    05 ECI-WINDOW-HANDLE              PIC S9(8) COMP-5.
```

```
    05 ECI-SEM-HANDLE REDEFINES ECI-WINDOW-HANDLE
                                PIC S9(8) COMP-5.
    05 FILLER        REDEFINES ECI-WINDOW-HANDLE.
       10 ECI-MS-WINDOW-HANDLE          PIC S9(4) COMP-5.
       10 ECI-MS-INSTANCE-HANDLE        PIC S9(4) COMP-5.
    05 ECI-MESSAGE-ID                 PIC  9(4) COMP-5.
    05 ECI-MESSAGE-QUALIFIER          PIC S9(4) COMP-5.
    05 ECI-LUW-TOKEN                  PIC S9(8) COMP-5.
          88 ECI-LUW-NEW              VALUE 0.
    05 ECI-SYSID                   PIC X(4).
    05 ECI-VERSION                 PIC S9(4) COMP-5.
          88 ECI-VERSION-0            VALUE 0.
          88 ECI-VERSION-1            VALUE 1.
          88 ECI-VERSION-1A           VALUE 2.
          88 ECI-VERSION-MAX          VALUE 2.
    05 ECI-SYSTEM-NAME                PIC X(8).
    05 ECI-CALLBACK             PROCEDURE-POINTER.
    05 ECI-USERID2                 PIC X(16).
    05 ECI-PASSWORD2                  PIC X(16).
    05 ECI-TPN                      PIC X(4).
*    05 ECI-COMMAREA POINTER.
*
* List of error returns from CICSEXTERNALCALL
*

 01 ECI-ERROR-ID                      PIC S9(4) COMP-5.
    88 ECI-NO-ERROR                   VALUE  0.
    88 ECI-ERR-INVALID-DATA-LENGTH        VALUE -1.
    88 ECI-ERR-INVALID-EXTEND-MODE        VALUE -2.
    88 ECI-ERR-NO-CICS                 VALUE -3.
    88 ECI-ERR-CICS-DIED               VALUE -4.
    88 ECI-ERR-REQUEST-TIMEOUT           VALUE -5.
    88 ECI-ERR-NO-REPLY                VALUE -5.
    88 ECI-ERR-RESPONSE-TIMEOUT          VALUE -6.
    88 ECI-ERR-TRANSACTION-ABEND         VALUE -7.
    88 ECI-ERR-EXEC-NOT-RESIDENT         VALUE -8.
    88 ECI-ERR-LUW-TOKEN               VALUE -8.
    88 ECI-ERR-SYSTEM-ERROR              VALUE -9.
    88 ECI-ERR-NULL-WIN-HANDLE           VALUE -10.
    88 ECI-ERR-NULL-MESSAGE-ID           VALUE -12.
    88 ECI-ERR-THREAD-CREATE-ERROR        VALUE -13.
    88 ECI-ERR-INVALID-CALL-TYPE         VALUE -14.
    88 ECI-ERR-ALREADY-ACTIVE            VALUE -15.
    88 ECI-ERR-RESOURCE-SHORTAGE         VALUE -16.
    88 ECI-ERR-NO-SESSIONS             VALUE -17.
    88 ECI-ERR-NULL-SEM-HANDLE           VALUE -18.
    88 ECI-ERR-INVALID-DATA-AREA         VALUE -19.
    88 ECI-ERR-INVALID-VERSION           VALUE -21.
    88 ECI-ERR-UNKNOWN-SERVER            VALUE -22.
    88 ECI-ERR-CALL-FROM-CALLBACK        VALUE -23.
    88 ECI-ERR-INVALID-TRANSID           VALUE -24.
    88 ECI-ERR-MORE-SYSTEMS              VALUE -25.
    88 ECI-ERR-NO-SYSTEMS              VALUE -26.
    88 ECI-ERR-SECURITY-ERROR            VALUE -27.
    88 ECI-ERR-MAX-SYSTEMS             VALUE -28.
    88 ECI-ERR-MAX-SESSIONS            VALUE -29.
    88 ECI-ERR-ROLLEDBACK              VALUE -30.

*
* Commarea layout for ECI-STATE-xxx CallType requests other
* than when the  ExtendMode is ECI-STATE-CANCEL.
*
* It should be supplied with valid values for a request where
```

```
* the ExtendMode is ECI-STATE-CHANGED.  In this case a
* response will be returned only when the status is
* different to that which was supplied.
*
  * It will be returned with the current status in these fields
  * except where  the ExtendMode is ECI-STATE-CANCEL.

 01 ECI-STATUS.
    05 ECI-CONNECTION-TYPE              PIC S9(4) COMP-5.
          88 ECI-CONNECTED-NOWHERE        VALUE 0.
          88 ECI-CONNECTED-TO-SERVER      VALUE 1.
          88 ECI-CONNECTED-TO-CLIENT      VALUE 2.
    05 ECI-CICS-SERVER-STATUS           PIC S9(4) COMP-5.
          88 ECI-SERVERSTATE-UNKNOWN      VALUE 0.
          88 ECI-SERVERSTATE-UP           VALUE 1.
          88 ECI-SERVERSTATE-DOWN         VALUE 2.
    05 ECI-CICS-CLIENT-STATUS           PIC S9(4) COMP-5.
          88 ECI-CLIENTSTATE-UNKNOWN      VALUE 0.
          88 ECI-CLIENTSTATE-UP           VALUE 1.
          88 ECI-CLIENTSTATE-INAPPLICABLE  VALUE 2.
    05 FILLER                           PIC S9(4) COMP-5.
    05 FILLER                           PIC S9(4) COMP-5.
*  01 COMMAREA PIC X(300).
*
*  CICSECILISTSYSTEMS.
*
*  Note: The value '16' assigned to CICS-ECINUMSYS and the X
*  except value in the OCCURS clause for CICS-ECISYSTEM may
*  need be increased if the ECI-ERR-MORE-SYSTEMS error occurs.
*

 77 CICS-ECI-SYSTEM-MAX          PIC 9(4) COMP-5 VALUE 8.
 77 CICS-ECI-DESCRIPTION-MAX     PIC 9(4) COMP-5 VALUE 60.

 77 CICS-ECINUMSYS               PIC 9(4) COMP-5 VALUE 16.

 01 CICS-ECISYSTEM.
    02 FILLER
    OCCURS 0 TO 16 TIMES DEPENDING ON CICS-ECINUMSYS.
       05 SYSTEMNAME           PIC X(8).
       05 FILLER               PIC X.
       05 SYSTEMDESC           PIC X(60).
       05 FILLER               PIC X.


 01 COMMAREA2 PIC X(500).
 LINKAGE SECTION.
 01 COMMAREA.
   05 SEND-DATA PIC X(147).
   05 MSG-QUAL PIC X(3).
   05 REC-DATA PIC X(150).
 PROCEDURE DIVISION USING COMMAREA.
*PROCEDURE DIVISION.
 P0.

      MOVE "This is the commarea for cics." TO COMMAREA.
*     MOVE "This is the commarea for cics." TO COMMAREA2.

*1      MOVE "DFHZCN1" TO ECI-PROGRAM-NAME.
*1      MOVE "CCIN"  TO ECI-TRANSID.
*1      MOVE "CPMI" TO ECI-TPN.
```

```
*2      MOVE "DFHZCN1" TO ECI-PROGRAM-NAME.
*2      MOVE "CCIN"  TO ECI-TRANSID.
*2      MOVE "CPMI" TO ECI-TPN.

       MOVE "LINKPROG" TO ECI-PROGRAM-NAME.
       MOVE "CPMI"  TO ECI-TRANSID.
*       MOVE "CPMI" TO ECI-TPN.

       MOVE "P390" TO  ECI-USERID.
       MOVE "IBM7TED" TO ECI-PASSWORD.
*
       MOVE "aaaa"    TO ECI-ABEND-CODE.
       SET ECI-COMMAREA TO ADDRESS OF COMMAREA
*      SET  ADDRESS OF COMMAREA TO ECI-COMMAREA.
       MOVE  LENGTH  OF COMMAREA TO ECI-COMMAREA-LENGTH.
       DISPLAY "COBOL.COMMAREA.LENGTH=" ECI-COMMAREA-LENGTH.
       MOVE 00    TO ECI-TIMEOUT.
       MOVE ZERO    TO ECI-SYS-RETURN-CODE.
       MOVE ZERO   TO ECI-EXTEND-MODE.
       MOVE ZERO   TO ECI-WINDOW-HANDLE.
       MOVE ZERO   TO ECI-SEM-HANDLE.
       MOVE ZERO   TO ECI-MESSAGE-ID.
       MOVE ZERO   TO ECI-MESSAGE-QUALIFIER.
       MOVE ZERO   TO ECI-LUW-TOKEN.
       MOVE "sys1" TO ECI-SYSID.
       MOVE 1   TO ECI-VERSION.
       MOVE "P390" TO ECI-SYSTEM-NAME.
       MOVE 1      TO CICS-ECI-SYSTEM-MAX.
*      SET   ECI-CALLBACK TO  ENTRY P0.
       SET   ECI-CALLBACK TO  NULL.

       MOVE "P390SNA " TO ECI-SYSTEM-NAME.
       MOVE "user2 " TO ECI-USERID2.
       MOVE "ps2 " TO ECI-PASSWORD2.

       MOVE "This is the commarea for cics ECI-ASYNC."
            TO SEND-DATA.
       MOVE " " TO REC-DATA.
       SET ECI-ASYNC                 TO TRUE.
       MOVE 1   TO ECI-MESSAGE-QUALIFIER.
       MOVE "001" TO MSG-QUAL.
       CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
       DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
       DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
       DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

       MOVE "This is the commarea for cics ECI-ASYNC."
            TO SEND-DATA.
       MOVE " " TO REC-DATA.
       SET ECI-ASYNC                 TO TRUE.
       MOVE 2   TO ECI-MESSAGE-QUALIFIER.
       MOVE "002" TO MSG-QUAL.
       MOVE ZERO   TO ECI-LUW-TOKEN.
       SET ECI-LUW-NEW TO TRUE.
       CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
       DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
       DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
       DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

       MOVE "This is the commarea for cics ECI-ASYNC."
            TO SEND-DATA.
       MOVE " " TO REC-DATA.
       SET ECI-ASYNC                 TO TRUE.
```

```
           MOVE 3   TO ECI-MESSAGE-QUALIFIER.
           MOVE "003" TO MSG-QUAL.
           MOVE ZERO   TO ECI-LUW-TOKEN.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

           MOVE "This is the commarea for cics ECI-ASYNC."
                TO SEND-DATA.
           MOVE " " TO REC-DATA.
           SET ECI-ASYNC                 TO TRUE.
           MOVE 4   TO ECI-MESSAGE-QUALIFIER.
           MOVE "004" TO MSG-QUAL.
           MOVE ZERO   TO ECI-LUW-TOKEN.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

           MOVE "This is the commarea for cics ECI-ASYNC."
                TO SEND-DATA.
           MOVE " " TO REC-DATA.
           SET ECI-ASYNC                 TO TRUE.
           MOVE 5   TO ECI-MESSAGE-QUALIFIER.
           MOVE "005" TO MSG-QUAL.
           MOVE ZERO   TO ECI-LUW-TOKEN.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

           MOVE "This is the commarea for cics ECI-ASYNC."
                TO SEND-DATA.
           MOVE " " TO REC-DATA.
           SET ECI-ASYNC                 TO TRUE.
           MOVE 6   TO ECI-MESSAGE-QUALIFIER.
           MOVE "006" TO MSG-QUAL.
           MOVE ZERO   TO ECI-LUW-TOKEN.
           CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
           DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
           DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
           DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

          SET ECI-CALLBACK TO ENTRY P1.
          MOVE "This is the commarea for cics ECI-GET-REPLY."
               TO SEND-DATA.
          MOVE " " TO REC-DATA.
          DISPLAY "COMMAREA " COMMAREA.
          SET ECI-GET-SPECIFIC-REPLY          TO TRUE.
          MOVE 3 TO ECI-MESSAGE-QUALIFIER.
          MOVE ZERO   TO ECI-LUW-TOKEN.
          CALL 'CICSEXTERNALCALL' USING ECI-PARMS.
          DISPLAY "ECI-SYS-RET-Code= " ECI-SYS-RETURN-CODE.
          DISPLAY "ECI-ABEND-CODE= " ECI-ABEND-CODE.
          DISPLAY "Commarea returned from P390/CICS= " COMMAREA.

        STOP RUN.
      P1.
          MOVE "ABC" TO D.
          DISPLAY " P1 entered ".
```

IMS Client

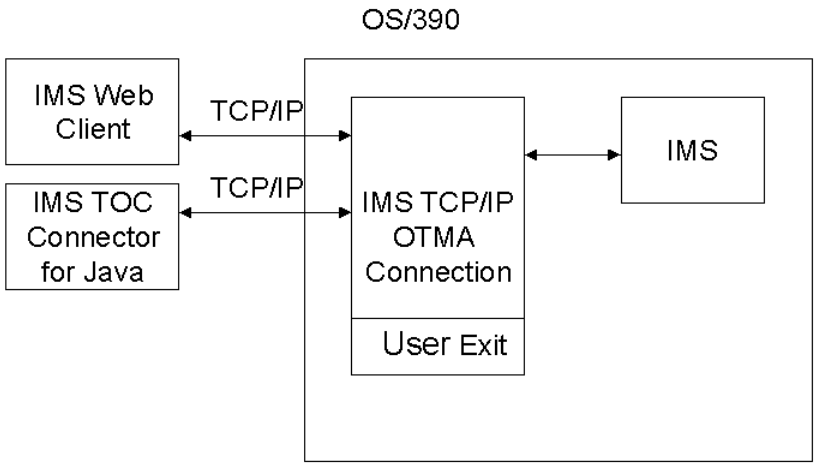
Requirements

1. OTMA installed and configured on OS390 using IMS TCPIP OTMA Connection User's Guide and Reference, ITOCUG0021-3 from IBM.
2. TCPIP connection to S/390

Overview

The IMS TCP/IP OTMA Connection (IMS TOC) is a TCP/IP server that enables TCP/IP clients to exchange messages with IMS OTMA. This server provides communication linkages between TCP/IP clients and IMS (data stores) on the OS/390 platform. Response messages from the IMS (data stores) are passed back to the originating TCP/IP clients.

COBOL programs using the IMS protocol in Elastic COBOL use the IMS TOC Connector for Java™ to send and receive data from IMS (data stores).



Elastic COBOL IMS Client support uses a file "interface" to send and receive data from IMS. An IMS "file" is opened I-O and uses READ/WRITE to send or receive IMS transaction data. A data record sent to IMS is dependent on the OTMA exit routine used to process the input/output to or from IMS.

The COBOL Client WRITEs to IMS contain all of the transaction input in one "record". COBOL Client READs contain only one IMS message segment per record. Additional COBOL Client READs are required to receive the entire transaction output.

An example of a COBOL program using the HWSSMPL0 exit is provided for reference. HWSSMPL0 is an OTMA supplied exit.

OTMA configuration file

```
************************************************
* IMS TOC EXAMPLE CONFIGURATION FILE
************************************************
HWS (ID=HWS1,RACF=N)
TCPIP (HOSTNAME=P390,RACFID=P390,PORTID=(9999),EXIT=(HWSSMPL0))
DATASTORE
(ID=TEDS,GROUP=IMSGROUP,MEMBER=TEDSMEM,TMEMBER=IMS61CR1)
Program Example
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID.
000300     IMSCLIENT.
004100 ENVIRONMENT DIVISION.
004200 CONFIGURATION SECTION.
004300 SOURCE-COMPUTER.
004400     XXXXX082.
004500 OBJECT-COMPUTER.
004600     XXXXX083.
004700*
004800 INPUT-OUTPUT SECTION.
004900 FILE-CONTROL.
005900     SELECT SQ-FS1
006000           ACCESS MODE IS SEQUENTIAL
006100           SEQUENTIAL
006200           ASSIGN TO
006300     "ims://p390:9999/"
052410      FILE STATUS IS FILE-FS1-STAT
006400           .
006500*
006600*
006700 DATA DIVISION.
006800 FILE SECTION.
009200 FD  SQ-FS1
009400             .
009500 01  IMS-OUTPUT.
        05 IC-HEADER.
          15 IC-TOTAL-LENGTH          PIC S9(5) COMP-5.
        05 IC-PREFIX.
          15 IC-PREFIX-LENGTH         PIC S9(2) COMP-5.
          15 IC-FLAGS-ZZ              PIC 9(2) BINARY VALUE 0.
          15 IC-IDENTIFIER            PIC X(8).
          15 IC-RESERV1               PIC 9(8) BINARY VALUE 0.
          15 IC-RESERV2               PIC 9(8) BINARY VALUE 0.
          15 IC-CLIENT-ID             PIC X(8).
          15 IC-FLAG1                 PIC X(1)   VALUE 0.
          15 IC-COMMIT-MODE           PIC X(1)   VALUE 0.
```

```
           15 IC-SYNC-LEVEL              PIC X(1)   VALUE 0.
           15 IC-RESPONSE                PIC X(1)   VALUE 0.
           15 IC-TRANSACTION-CODE         PIC X(8).
           15 IC-DATASTORE-ID             PIC X(8).
           15 IC-LTERM-NAME               PIC X(8).
           15 IC-RACF-USERID              PIC X(8).
           15 IC-RACF-GROUP-NAME          PIC X(8).
           15 IC-PASSWORD                 PIC X(8).
         05 IC-OUTPUT-DATA.
           15 IC-OUTPUT-LENGTH          PIC S9(2) COMP-5.
           15 IC-OUTPUT-FLAG1           PIC X(1) VALUE "0".
           15 IC-OUTPUT-FLAG2           PIC X(1) VALUE "0".
           15 IC-OUTPUT-DATA-SEG        PIC X(240).
         05 IC-END-OUTPUT.
           15 IC-LAST-SEG              PIC 9(2) BINARY VALUE 4.
           15 IC-LAST-FLAG             PIC 9(2) BINARY VALUE 0.
009400               .
009700 WORKING-STORAGE SECTION.
052410   01 FILE-FS1-STAT PIC XX.
009500   01  HEADER-CNT PIC 9(5) COMP-5.
009500   01  PREFIX-CNT PIC 9(5) COMP-5.
009500   01  OUTPUT-CNT PIC 9(5) COMP-5.
009500   01  LSTSEG-CNT PIC 9(5) COMP-5.
009500   01  LOOP-CNT   PIC 9(5) COMP-5 VALUE 0.
009500 01  IMS-INPUT2.
009500    05 IN-LTH  PIC 9(2) COMP-5.
         05 IN-FLAGS-ZZ                PIC 9(2) BINARY VALUE 0.
009500    05 IN-DATA PIC X(80).
009500 01  IMS-INPUT.
009500    05 REC-LTH  PIC 9(2) COMP-5.
         05 REC-FLAGS-ZZ               PIC 9(2) BINARY VALUE 0.
009500    05 REC-DATA PIC X(32700).
009600*
      01  PG-END-OUTPUT.
         05 PG-LAST-SEG              PIC 9(2) BINARY VALUE 4.
         05 PG-LAST-FLAG             PIC 9(2) BINARY VALUE 0.
      01  PART-NUMS.
         05 PART-N PIC X(30) OCCURS 5 TIMES.
      01  TRANSACTION-PATTERN.
         05 TRANSACTION PIC X(8) VALUE "PART ".
         05 PART-NUMBER PIC X(30).
009600*
031100 PROCEDURE DIVISION.
031200*
031300 CCVS1 SECTION.
         MOVE 0 TO LOOP-CNT.
         MOVE "3003802" TO PART-N(1).
         MOVE "7736847P001" TO PART-N(2).
         MOVE "7630843P513" TO PART-N(3).
         MOVE "930331-102" TO PART-N(4).
         MOVE "60003-118" TO PART-N(5).
      LOOP0.
         DISPLAY "LOOP-CNT= " LOOP-CNT.
051900     OPEN I-O    SQ-FS1.
         DISPLAY "LOOP-CNT= " LOOP-CNT.
         ADD 1 TO LOOP-CNT.
         DISPLAY "Part-N(" LOOP-CNT ")= " PART-N(LOOP-CNT).
         MOVE LENGTH OF IC-HEADER TO HEADER-CNT.
         DISPLAY "Header-cnt= " HEADER-CNT.
         MOVE LENGTH OF IC-PREFIX TO PREFIX-CNT.
         DISPLAY "Prefix-cnt= " PREFIX-CNT.
         MOVE LENGTH OF IC-OUTPUT-DATA TO OUTPUT-CNT.
         DISPLAY "Output-cnt= " OUTPUT-CNT.
```

```
           MOVE LENGTH OF IC-END-OUTPUT TO LSTSEG-CNT.
           DISPLAY "LSTSEG-CNT= " LSTSEG-CNT.
           MOVE 0 TO IC-TOTAL-LENGTH.
           ADD  HEADER-CNT TO IC-TOTAL-LENGTH.
           DISPLAY "IC-TOTAL-LENGTH= " IC-TOTAL-LENGTH.
           ADD  PREFIX-CNT TO IC-TOTAL-LENGTH.
           DISPLAY "IC-TOTAL-LENGTH= " IC-TOTAL-LENGTH.
           ADD  OUTPUT-CNT TO IC-TOTAL-LENGTH.
           DISPLAY "IC-TOTAL-LENGTH= " IC-TOTAL-LENGTH.
           ADD  LSTSEG-CNT TO IC-TOTAL-LENGTH.
           DISPLAY "IC-TOTAL-LENGTH= " IC-TOTAL-LENGTH.
           MOVE PREFIX-CNT TO IC-PREFIX-LENGTH.
           MOVE 0  TO IC-FLAGS-ZZ.
           MOVE "*SAMPLE*" TO IC-IDENTIFIER.
           MOVE FUNCTION CHAR(1) TO IC-RESERV1.
           MOVE FUNCTION CHAR(1) TO IC-RESERV2.
           MOVE "WARPED" TO IC-CLIENT-ID.
           MOVE FUNCTION CHAR(1) TO IC-FLAG1.
           MOVE FUNCTION CHAR(1) TO IC-COMMIT-MODE.
           MOVE FUNCTION CHAR(1) TO IC-SYNC-LEVEL.
           MOVE FUNCTION CHAR(1) TO IC-RESPONSE .
           MOVE "PART" TO IC-TRANSACTION-CODE .
           MOVE "TEDS" TO IC-DATASTORE-ID.
           MOVE "LTERMNAME" TO IC-LTERM-NAME.
           MOVE "P390" TO IC-RACF-USERID.
           MOVE "SYS1" TO IC-RACF-GROUP-NAME.
           MOVE "PASSWORD" TO IC-PASSWORD.
           MOVE OUTPUT-CNT TO IC-OUTPUT-LENGTH.
           MOVE FUNCTION CHAR(1) TO IC-OUTPUT-FLAG1.
           MOVE FUNCTION CHAR(1) TO IC-OUTPUT-FLAG2.
           MOVE PG-LAST-SEG        TO IC-LAST-SEG.
           MOVE PG-LAST-FLAG       TO IC-LAST-FLAG.
           MOVE PART-N(LOOP-CNT) TO PART-NUMBER.
           MOVE TRANSACTION-PATTERN TO IC-OUTPUT-DATA-SEG.
           DISPLAY "IC-TOTAL-LENGTH= " IC-TOTAL-LENGTH.
           DISPLAY "IC-PREFIX-LENGTH= " IC-PREFIX-LENGTH.
           DISPLAY "IC-FLAGS-ZZ= " IC-FLAGS-ZZ.
           DISPLAY "IC-IDENTIFIER= " IC-IDENTIFIER.
           DISPLAY "IC-RESERV1= " IC-RESERV1.
           DISPLAY "IC-RESERV2= " IC-RESERV2.
           DISPLAY "IC-CLIENT-ID= " IC-CLIENT-ID.
           DISPLAY "IC-COMMIT-MODE= " IC-COMMIT-MODE.
           DISPLAY "IC-SYNC-LEVEL= " IC-SYNC-LEVEL.
           DISPLAY "IC-RESPONSE= " IC-RESPONSE.
           DISPLAY "IC-TRANSACTION-CODE= " IC-TRANSACTION-CODE.
           DISPLAY "IC-DATASTORE-ID= " IC-DATASTORE-ID.
           DISPLAY "IC-LTERM-NAME= " IC-LTERM-NAME.
           DISPLAY "IC-RACF-USERID= " IC-RACF-USERID.
           DISPLAY "IC-RACF-GROUP-NAME= " IC-RACF-GROUP-NAME.
           DISPLAY "IC-PASSWORD= " IC-PASSWORD.
           DISPLAY "IC-OUTPUT-LENGTH= " IC-OUTPUT-LENGTH.
           DISPLAY "IC-OUTPUT-FLAG1= " IC-OUTPUT-FLAG1.
           DISPLAY "IC-OUTPUT-FLAG2= " IC-OUTPUT-FLAG2.
           DISPLAY "IC-OUTPUT-DATA-SEG= " IC-OUTPUT-DATA-SEG.
           DISPLAY "IC-LAST-SEG= " IC-LAST-SEG.
           DISPLAY "IC-LAST-FLAG= " IC-LAST-FLAG.
           DISPLAY "end of ims-otma     " .
           DISPLAY "lth of IC-HEADER= " LENGTH OF IC-HEADER.
           DISPLAY "lth of IC-OUTPUT-DATA= " LENGTH OF IC-OUTPUT-DATA.
           DISPLAY "lth of IC-END-OUTPUT= " LENGTH OF IC-END-OUTPUT.
           DISPLAY "LOOP-CNT= " LOOP-CNT.
       052400     WRITE   IMS-OUTPUT.
           DISPLAY "LOOP-CNT= " LOOP-CNT.
```

```
052400     MOVE " " TO IMS-OUTPUT.
           MOVE    " " TO IMS-INPUT.
           MOVE    80  TO REC-LTH
           MOVE    FUNCTION CHAR(2) TO REC-FLAGS-ZZ.

         LOOP.
           DISPLAY "LOOP-CNT= " LOOP-CNT.
052400     READ    SQ-FS1 INTO IMS-INPUT AT END GO TO END-PGM.
           DISPLAY "LOOP-CNT= " LOOP-CNT.
052400     MOVE    IMS-INPUT TO IMS-INPUT2.
052400     DISPLAY "IN-LTH= " IN-LTH  " IN-FLAGS-ZZ= " IN-FLAGS-ZZ.
052400     DISPLAY "IN-DATA= " IN-DATA.
           GO TO LOOP.
         END-PGM.
052800     CLOSE   SQ-FS1.
           DISPLAY "LOOP-CNT= " LOOP-CNT.
           IF LOOP-CNT > 4 GO TO CLS-PGM.
           GO TO LOOP0.
         CLS-PGM.
```

Chapter 7 – Client/Server Execution

Overview

Most COBOL systems have only a single execution environment, the application. There are additional environments, though, officially and unofficially recognized, ranging from running within the mainframe CICS environment to running within a remote login.

Elastic COBOL has its own execution environments, environments that provide different capabilities to the contained program than the typical application. As Elastic COBOL programs are Java programs for execution purposes, several environments for Java programs work with Elastic COBOL programs as well.

Most of these alternate execution environments are for the server, with the notable exception of applets. They may require additional software to setup and execute, but they may also provide additional functionality to the contained COBOL program.

In all of these execution environments, applications are generally not simply executed, but rather the application is first deployed. Deployment allows the application, Elastic COBOL runtime, and media resources to be merged into a single file for simpler distribution and assignment of resources.

Choose the environment based upon need. Many applications may be executed as simple applications; do not make the program more complicated than necessary. If the program needs to expose a rich user interface and operate on the client without installation, then an Applet is appropriate. If the program needs to output HTML to the user, process web forms, needs a lightweight user interface, needs a quick startup, then it is likely a Servlet candidate. A CGI program is like a Servlet, but for those web servers without Servlet support. Enterprise JavaBeans, or EJB, is for those programs that have complex or highly demanding requirements including transaction support, distribution and other advanced infrastructure; EJBs often use Servlets as their user interface.

Applet

An Applet is a program running within a web browser, or started directly from a web browser. Normally, its main screen is within the browser's page itself. It will be started automatically when the end-user browses to the containing page, and it will be stopped automatically when the end-user browses away.

An Applet is downloaded automatically from the server, executing according to instructions found in the containing HTML page. After downloading, it executes on the clients browsing machine itself rather than the server.

Because the applet is executing on the client itself, it has several advantages and disadvantages particular to the execution environment.

While on the client, it has very fast response times to commands which do not need the network; unlike HTML which may require the server to handle data validation, such work may be done on the client without involvement of the server. It can expose a rich user interface, complete with graphical screen elements, to the end user. It can access sockets, files and database connections back on the server.

However, since it's running on the client, there are security restrictions about the actions it can do to the client's computer. It cannot directly access files, for reading, writing or deleting. It cannot access sockets, files or database connections aside from the server from which it was downloaded. It cannot access most system properties (environment variable equivalents). It cannot even print to the printer without permission from the end user. These restrictions may be reduced or eliminated by signing the applet with a digital signature to allow the end user to trust the applet, but signing is a rare requirement for an Applet. Generally, if the applet needs to be signed, it should probably be done through a server mechanism instead. No native code access is allowed from applets. All code must be in Elastic COBOL or Java.

The applet itself is served from the web server. It may be kept in the same directory as the HTML page containing the reference to the applet. During deployment, an option is given to create default HTML templates for starting the program as an applet; use the automatically generated HTML as a template, use its contents to incorporate in custom HTML pages designed by web designers.

The applet may run in a Java Virtual Machine (JVM) from the browser vendor, or it may operate within the context of a Java Plug-In, an add-on to the browser that provides consistent execution independent of the browser's implementation. The Java Plug-In provides a current environment, whereas the browser environments may become dated. For example, the graphical screen section requires Java 2 (JDK 1.2) support, so it requires the Java Plug-In to function as an Applet. Note: Netscape 6 and above use the Java Plug-In for all applet support.

An applet during execution normally starts at the same point in the program as an application. Its main window, rather than being a separate window, is embedded within the HTML page. The size of the main window is set by the applet tags rather than the program, as HTML sets the boundaries for inline applets.

There is an alternative lifecycle for applets that allows the applet to recognize when the applet is created, started, stopped and finally destroyed. This usage is now discouraged where not required. If present, the APPLET-INIT, APPLET-START, APPLET-STOP and APPLET-DESTROY paragraphs will be executed at the appropriate point in the applet's lifecycle.

SYSOUT and SYSERR are routed to the browser's Java console window and should be used only for debugging or logging information, never for information vital to the user. SYSIN is not available. The CONSOLE will function normally in its graphical mode.

Extra frames created may have notification in them that the frame is a Java window to allow the user to recognize such windows as originating online; this helps prevent spoofing the actions of a legitimate program.

Servlet

Servlets run within the context of a web server, with the display run within the context of the end-user's web browser.

The execution is in a controlled environment, so most normal application abilities, such as file handling, are present. The execution environment itself is as secure as the surrounding operating system and environment, so business processes are not exposed to the end user.

The only portion of a Servlet exposed to the end user is the user interface, generally HTML. The program is started from HTML, reads its input from HTML forms, and sends its output back to the user in the form of HTML.

The program itself starts when requested to do so by a URL pointing to the Servlet, using either the GET or PUT method of data transfer. The web server must be Servlet enabled in order to do this; see the web server documentation for information on where to place Servlets for automatic activation.

Once the program starts, it reads its information from the form, processes it, and sends information back in the form of another HTML page. It does so at the same time other instances of the same program, and other programs, are also operating within the same process. Each instance that the web server creates is its own session with its own data.

This section does not discuss HTML itself, so a basic knowledge of HTML is assumed. The COBOL programmer creating Servlets will often do so in conjunction with a web designer for the HTML content. The goal of the HTML portions of this is to provide enough information to bridge the gap from the COBOL programmer's experience to the web designer's experience.

Compiling

The Servlet API is a standard extension to Java, which means that although its usage and interfaces are standardized, it is not included by default with Java. So, the Servlet runtime library must be present in the CLASSPATH as well as the normal Elastic COBOL runtime and application classes. This may already be done in the web server, but for test environments either the Servlet runtime servlet.jar or jsdk.jar must be present.

A Servlet may be compiled with a special flag, -out:servlet, so the main program is a Servlet itself. The Servlet classes must be in the classpath for this to function. Or it may be run indirectly, using com.heirloomcomputing.ecs.run.servlet as the main program, with an init parameter 'program' pointing to the main application class; this method requires only the Servlet runtime to execute.

Executing

The HTML to execute the Servlet must contain a tag pointing to the program. An example pointing to the local machine for the program register follows:

```
<form action="http://127.0.0.1:8080/servlet/register" method="post">
```

A GET action is the default, and can be performed even from the browser's URL line. For example, if the program 'simple' were on the host myhost.com, then typing 'http://www.myhost.com/servlet/simple' would activate the Servlet.

A program may either be a traditional program, where there is only one control path for all forms of execution, or it may use special paragraph names to indicate when each portion of code should be executed. The more common special paragraph names are as follows:

SERVLET-INIT	This is called only once, when the Servlet is initially loaded.
SERVLET	This is called for either GET or POST transactions.
SERVLET-GET	This is called for GET method transactions.
SERVLET-POST	This is called for POST method transaction.

If there is no application reason to distinguish between GET and POST, accepting either offers more flexibility for the web designer and testing.

Some additional less commonly used special paragraph names are also available. Most programmers will not require the use of these paragraphs, but they are available for completeness of information.

SERVLET-PUT	This activates only when the browser sends a PUT message.
SERVLET-DELETE	This activates only when the browser sends a DELETE message.
SERVLET-OPTIONS	This activates only when the browser sends an OPTIONS message.
SERVLET-TRACE	This activates only when the browser sends a TRACE message.
SERVLET-LAST-MODIFIED	Return the last known modification time in RETURN-CODE. This is handled automatically, so override only if necessary.

Retrieving User Data

Once the program is executing, all but the simplest output program will require data from the user. It may be something as simple as a zip code, or as complex as a shopping cart order. This data will come through the HTML form and will already be present at the time the program starts to execute.

The data may be entirely processed before data is sent back to the user, so interactions are spread between multiple Servlets. The HTML generated by

a Servlet may always point to another Servlet for next execution, the conceptual 'Next' on the HTML page.

Sending Response Data

In Elastic COBOL, the normal display device used for the DISPLAY statement without an UPON clause is the graphical console on graphical systems or the text terminal on text systems. In a Servlet, however, the default display device when no UPON clause is present is the browser's HTML input stream. So a DISPLAY "Hello World" will be displayed in the browser, as will a DISPLAY "Hello World" UPON SERVLET-OUT. A DISPLAY "Hello World" UPON CONSOLE will work in many cases, but will not have the intended effect; it would go to the server's version of the console. A DISPLAY UPON SYSOUT or SYSERR will generally be redirected to a web server or Servlet runner log file, and so they are suitable for logging server messages.

These responses are stored in the configuration space for the program and may be retrieved in the same manner as configuration properties, system properties, and specified environment variables. The simplest method to retrieve these responses is ACCEPT identifier FROM CONFIGURATION configuration-name. The configuration-name may include additional data in the name, such as parameter names from the form. In a similar manner, the modifiable configuration-names may be set using DISPLAY value UPON CONFIGURATION configuration-name.

All special configuration-names for Servlets begin with either SERVLET- or HTTP-. The following configuration-names are recognized only in Servlets.

Servlet Get Configuration Names	Description
SERVLET-PARAMETER-x	Obtain parameter x of request; x is generally a form variable name.
SERVLET-SESSION-PARAMETER-x	Obtain session parameter x of request; used to store and retrieve session persistent info.
SERVLET-QUERY-STRING	Obtain the query string used to obtain this Servlet.
SERVLET-COOKIES	Count of number of cookies available.
SERVLET-COOKIE-x	Obtain cookie number x.
SERVLET-ATTRIBUTE-x	Obtain value of named attribute x.
SERVLET-METHOD	Obtain HTTP method for this request, generally GET or POST.
SERVLET-REQUEST-URI	Obtain part of request's URI to left of query string.
SERVLET-PATH	Obtain part of request URI specifying the Servlet.
SERVLET-PATH-INFO	Obtain optional extra path info before query string.
SERVLET-PATH-TRANSLATED	Obtain optional extra path info before query string, translated to a real path.
SERVLET-REMOTE-USER	Obtain user name generating this request.
SERVLET-AUTH-TYPE	Obtain authentication scheme for this request.
SERVLET-HEADER-x	Obtain header x of request.
SERVLET-REQUESTED-SESSION-ID	Obtain identifier for this session.
SERVLET-REQUESTED-SESSION-ID-VALID	Obtain 1 if session is valid, 0 if invalid.
SERVLET-REQUESTED-SESSION-ID-FROM-COOKIE	Obtain 1 if request's session id came from cookie, 0 if not.
SERVLET-REQUESTED-SESSION-	Obtain 1 if request's session id came from URL, 0 if

Servlet Get Configuration Names	Description
ID-FROM-URL	not.
SERVLET-CONTENT-LENGTH	Obtain size of data, -1 if unknown.
SERVLET-CONTENT-TYPE	Obtain mime type of request data.
SERVLET-PROTOCOL	Obtain the <protocol>/<major>.<minor> of the request.
SERVLET-SCHEME	Obtain the protocol scheme, such as http or https.
SERVLET-SERVER-NAME	Obtain the host name of the server.
SERVLET-SERVER-PORT	Obtain the host port of the server.
SERVLET-REMOTE-ADDR	Obtain address of the client.
SERVLET-REMOTE-HOST	Obtain host name of the client.
SERVLET-CHARACTER-ENCODING	Obtain character set encoding for input of request.
SERVLET-SESSION-CREATION-TIME	Obtain time session was created.
SERVLET-SESSION-ID	Obtain serve's session id.
SERVLET-SESSION-LAST-ACCESSED-TIME	Obtain last time session was accessed.
SERVLET-SESSION-NEW	Obtain 0 if session is new session, 1 if client is already bound.
SERVLET-CONTAINS-HEADER-x	Obtain 1 is Servlet response contains header x, 0 otherwise.
SERVLET-SESSION-MAX-INACTIVE-INTERVAL	Obtain maximum inactive time for session.
SERVLET-SESSION-ATTRIBUTE-x	Obtain session attribute x.
SERVLET-CONTAINS-HEADER-x	Obtain 1 if Servlet contains header x, 0 otherwise.
SERVLET-ATTRIBUTE-x	Obtain Servlet attribute x.
SERVLET-INIT-PARAMETER-x	Obtain value of Servlet initialization parameter x.
SERVLET-NAME	Obtain name of Servlet.

Servlet Set Configuration Names	Description
SERVLET-REDIRECT	Redirect user HTML page to value.
SERVLET-STATUS-x	Set status x with message value.
SERVLET-CONTENT-LENGTH	Set content length of HTML response.
SERVLET-CONTENT-TYPE	Set content type of HTML response.
SERVLET-SESSION-PARAMETER-x	Set session parameter to value.
SERLVET-SESSION-MAX-INACTIVE-INTERVAL	Set maximum inactive time for session.
SERVLET-SESSION-ATTRIBUTE-x	Set Servlet session attribute x to value.
SERVLET-SESSION-INVALIDATE	Invalid Servlet session.

EXEC HTML

There is a special EXEC command, EXEC HTML, for sending HTML to the browser's HTML input stream. In a Servlet, it will automatically use SERVLET-OUT, while in CGI discussed later, it will use SYSOUT. For more information on EXEC HTML, see HTML in the user interface chapter.

EXEC HTML supports direct embedding of HTML code into Elastic COBOL programs. There are two main formats to this extension.

Format 1:

```
EXEC HTML
Html-text-1
END-EXEC
```

Format 2:

```
EXEC PAGE-HTML
Html-text-1
END-EXEC
```

EXEC HTML, in both variants, outputs its *Html-text-1* to SYSOUT. This output is then processed by an HTTP web server, and sent to the client's browser window in HTML format. This must be properly formatted HTML in order to be received properly; no validation of HTML occurs. *Html-text-1* may span several lines.

Format 2 outputs the following text before *Html-text-1*:

```
Content-type: text/html
Content-length: length-of-html-text-1
<blank line>
```

This format 2 output is more proper for text pages and will automatically calculate the length of the text being sent. It cannot, however, be used to continue an HTML page already started by another EXEC HTML.

HOSTVAR tag

In addition to standard HTML, there is an extra HTML tag supported by Elastic COBOL, the HOSTVAR tag. The HOSTVAR tag replaces its tag with the contents of an execution host variable used prior to sending data to SYSOUT.

The format of the HOSTVAR tar is as follows:

```
<HOSTVAR NAME=host-variable-name>
text-skipped-by-ElasticCOBOL
</HOSTVAR>
```

Host-variable-name must be an alphanumeric, alphanumeric-edited, alphabetic, numeric, or numeric-edited identifier.

For proper viewing, numeric identifiers should be of USAGE DISPLAY. The HOSTVAR tag must be contained completely on one line.

The text-skipped-by-ElasticCOBOL will not be displayed when this HTML is executed through Elastic COBOL, but it will be displayed if the same HTML code is displayed through another means. It could, for example, include a warning that the HTML code is intended to be executed from Elastic COBOL.

The sample program html.cob shows the use of the EXEC HTML and HOSTVAR tag. A Web Server is not required to see its output.

CGI

CGI, or Common Gateway Interface, programs were the first common way to write programs capable of outputting HTML to the web and acting on the data input from forms. While Elastic COBOL programs may act as CGI programs, the usage is discouraged in favor of Servlets.

CGI requires a separate process for each transaction, with its own startup time, its own memory space, and its own tear down time. Servlets require only a thread for each transaction, with minimal startup, memory and tear down requirements.

CGI does not have the ability to readily save state. Servlets can readily save state from one execution to the next.

CGI programs do have the ability to execute in web server environments that do not support Servlets directly, but this time is almost always better spent in setting up the Servlet container for the web server instead and then executing the Elastic COBOL programs as Servlets.

CGI and Servlets use the same programming model. Both can output HTML to SERVLET-OUT, or use EXEC HTML to embed HTML content within the program. Both can access parameters from forms.

To start a program as a CGI program, setup the web server for CGI according to its directions. Then setup a batch or script file to execute the Elastic COBOL program using the following line:

Windows:

```
java -DREQUEST_METHOD="%REQUEST_METHOD%" -
DCONTENT_LENGTH="%CONTENT_LENGTH%" -
DQUERY_STRING="%QUERY_STRING%" -cp ecobol.jar;. AA
```

Posix:

```
java -DREQUEST_METHOD="$REQUEST_METHOD" -
DCONTENT_LENGTH="$CONTENT_LENGTH" -
DQUERY_STRING="$QUERY_STRING" -cp ecobol.jar:. AA
```

Instead of java, use whatever java command is required for the system. Specify the full path to ecobol.jar if necessary, or include any additional third-party runtime that may be required by your application. Replace AA with the application's program .class name, not including the .class extension; be sure to include the applications path in the classpath specified after -cp.

RMI

RMI is Remote Method Invocation. It is the standard mechanism by which remote procedure calls are done within the Java environment. When functions should be centrally defined or administered, RMI can offer a relatively simple means of exporting and using remote functionality.

Coverage of RMI as it relates to Elastic COBOL is found in Chapter 2 in COBOL to COBOL Remote, COBOL to Java Remote, and Java to COBOL Remote sections.

Enterprise JavaBeans

Elastic COBOL Enterprise JavaBeans involves programming constructs related to building Enterprise Java Beans. Elastic COBOL adds features that extend standard COBOL-85 and offers functional capabilities that are not defined as part of the COBOL standard.

Note: Heirloom's Elastic Transaction Platform will compile and generate Enterprise Java Beans from CICS COBOL programs.

Introduction

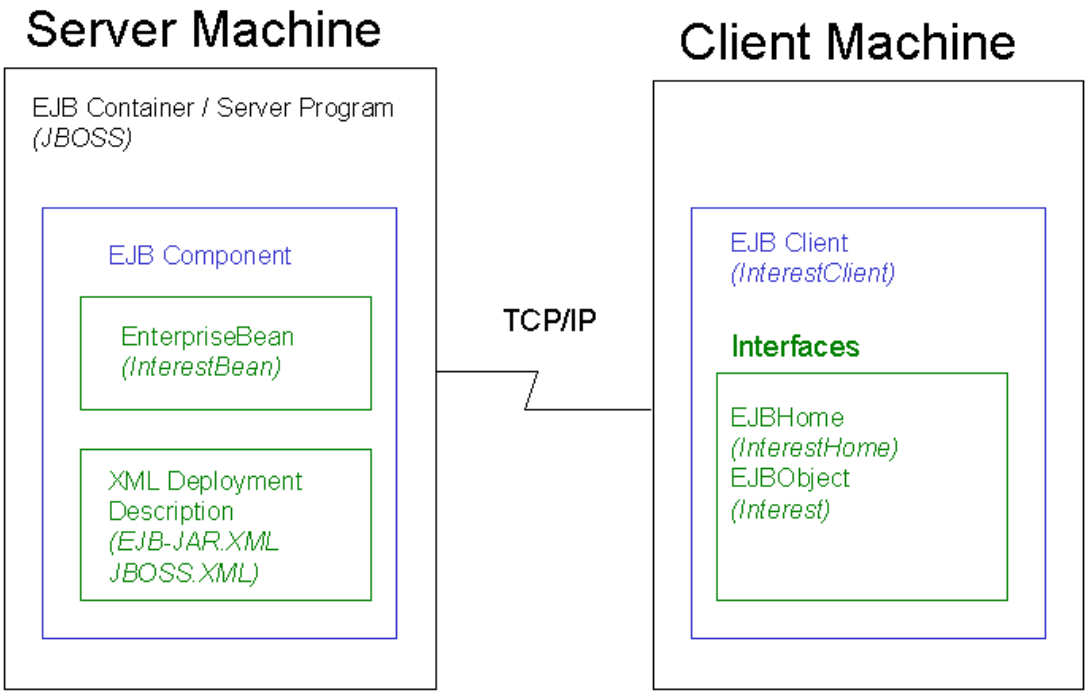
The EJB Container used in this sample is JBoss, a freely available EJB Container available at http://www.jboss.org. This Enterprise JavaBeans (EJB) Guide follows the JBoss tutorial found at http://www.jboss.org/documentation/jboss_win32_1.html.

EJB Terms

Term	Description	Example
EnterpriseBean	Contains the business logic	InterestBean
EJB Client	User of business logic	InterestClient
EJBObject	Client view of the EJB Component	Interest
EJBHome	Used by Client to obtain the EJBObject.	InterestHome
JNDI	Java naming and directory interfaces; used to obtain EJBHome	

Component Overview

EJB's are server side components in a standard format available for use by clients, which may include applications, applets, Servlets and other EJBs.



To execute the EJB sample, an EJB Container, EJB Component and EJB Client are required. The EJB Container used in this example is JBoss, a freely available EJB Container; other EJB Containers will be the same for coding, but may differ in setup and the values for the environment variables.

The EJB Container provides the distributed business object framework. It provides services to the EJB Component, allowing it to concentrate on the business functionality rather than the developing a custom distributed framework. An EJB Container can provide additional services such as transparently allowing a failed object to be replaced by a successful one. Because the Java platform is cross-platform, it can even allow transparent access of a bean to be split among different operating systems. There are many EJB Containers, each with its own set of strengths. Some concentrate on cross-platform issues, some on interoperability, some on wireless clients, etc.

The EJB Component provides the business functionality to remote clients. This EJB component may be written in Elastic COBOL or Java.

The EJB Client uses an EJB Component. The EJB Client may be written in Elastic COBOL or Java. If the EJB container supports it, it may also include additional access methods such as wireless protocols or COM.

Java Naming Directory Interface

Client Settings	Description
Java.naming.factory.initial	Class used to lookup EJBs by name
Java.naming.provider.url	Informs lookup class where to located naming service

EJB Execution Process

1. Copy EJB Component to server deploy directory
2. Start Server
3. Start Client

Install JBoss

Installation of JBoss follows the instructions available at the JBoss site for Win32 or Unix.

An additional note is that the Elastic COBOL runtime, ecobol.jar, must be found by EJB Container for Elastic COBOL EJB Components to function. This may be done in the deploy tool, which can combine an application such as an EJB Component with the Elastic COBOL runtime into a single deployable .jar file, or it may be done by adding ecobol.jar to the EJB Container's CLASSPATH. If added to the CLASSPATH of the EJB Container, the EJB Component .jar file does not need to include the contents of ecobol.jar; if not added to the CLASSPATH, then the deployment must combine ecobol.jar with the applications classes.

The EJB Container should execute normally before proceeding.

Creating the Enterprise JavaBean Component

In this step we will write and compile a simple Enterprise JavaBean. The example -- which is called 'Interest' -- is about as simple as an EJB can get: it is a 'stateless session bean'. Its job is to calculate the amount of compound interest payable on a sum of money borrowed over a specified term with a specified interest rate.

This example looks like it may have a number of lines of code, but most of it is purely structural, identical for all EJB Components created for a particular EJB Container. In fact, there is only one functional line of code in the whole package.

Create a new project in the IDE or directory for command line use. The directory '\ecobol\samples\ejb\server' will be used in this text.

Generally, EJB components are deployed in packages. A package is separate namespace for an application preventing names of programs from conflicting with one another. A package may be specified in one of three

ways; in the program source, in the IDE when compiling in the IDE, or on the command line when compiling from the command line.

In this case package will be 'com.web_tomorrow.interest'. This sample places the package name in the program source. It may be removed and done at compile time if desired. This means that Elastic COBOL will use the -package compiler option or package field in the Project|ecobol|COBOL Properties in the IDE.

An Enterprise JavaBean has a minimum of one class and two interfaces.


The EJB Component or Bean class. This implements the methods specified by the remote interface. In this example, the Bean class is 'com.web_tomorrow.interest.InterestBean'.


The remote interface. This is the class that exposes methods of the Bean to the outside world. In the example, the remote interface is the class 'com.web_tomorrow.interest.Interest'. Only the methods specified in this interface may be called from remote.


The home interface. This specifies how a new bean is created, managed and deleted. The methods in the home interface depend on the type of bean. A stateless session bean has exactly one create method. In this example, the home interface is 'com.web_tomorrow.InterestHome'.

Of course, a Bean can include other classes, other programs, or even other packages, but the classes listed above are the minimum. The classes must be packaged into a JAR, a Java Archive, with a directory structure which reflects the hierarchy of packages. The Elastic COBOL deploy tool may be used to create the actual JAR file, but the directory layout needs to be done first. In this example, the classes are in the package com.web_tomorrow.interest, so they need to be in the directory

\ecobol\samples\ejb\server\com\web_tomorrow_interest

Elastic COBOL will create this directory automatically from the COBOL source files if the package option is used. In Elastic COBOL, packages are created in the directory described by the package below the source file directory. There also needs to be a directory called META-INF to store the deployment descriptor (always called ejb-jar.xml) and, optionally, another XML file to tell the server about name mappings. With JBoss, this file must be called jboss.xml.

So, before writing the classes, we need a directory structure like this:

\ecobol\samples\ejb\server

.cob files are compiled here with -package option, yielding .java and .class files below

com

web_tomorrow

interest

.java and .class files are compiled to here

META-INF

ejb-jar.xml

jboss.xml

If the deploy tool has the \ecobol\samples\ejb\server directory files added, the directory structure will be complete. The visible files to be deployed should begin with com/ or META-INF/.

Of course, in a custom project rather than this sample project, the directory \ecobol\samples\ejb\server would be replaced with a custom directory structure representing the project and company. The standard for package names is the reverse of the company URL; that is if the web site is www.mycompany.com, then the project 'myproject' at that company would be in package com.mycompany.myproject

Coding the Classes and Interfaces

We need one class, the Bean itself, and two glue interfaces, the remote interface and the home interface. All the '.java' files are compiled into the subdirectory .\com\web_tomorrow\interest.

The interfaces have no method body, but rather only a definition of what methods are guaranteed to be present in any class implementing the interface. Using an interface is like using a class; the object is actually an instance of a class, but when accessing it via an interface, only the interface methods can be used.

In this case, the interface specifies the methods exposed to the EJB client, and the allowed method of creating the bean.

Remote Interface

The remote interface in this example is very simple.

```
* This example is coded for the JBoss EJB Container.  The naming
* provider and setup information may differ for other
* EJB Containers.
* The logic flow remains the same. (http://www.jboss.org)
*
* This example requires J2EE.  JDK 1.2+ plus JBoss is sufficient.
*
* The CLASSPATH to compile must include \jboss\lib\ext\ejb.jar in
* addition to the ecobol.jar required for any COBOL program.
*
* Set the package name; this could be done from the command line
* using the '-package name' directive instead.  Setting a package
* is generally not necessary for an EJB, but it's good practice.

$SET PACKAGE com.web_tomorrow.interest

*
* An INTERFACE-ID is the same as a CLASS-ID, but without any
* method definitions.  No special-names, no data other than
* linkage, and nothing in the procedure division other than
* the procedure division using is allowed.  This defines a
* contract which another class-id program may implement.
```

```
*

IDENTIFICATION DIVISION.
INTERFACE-ID. "Interest" INHERITS "javax.ejb.EJBObject".

* Define one method with three parameters which is capable
* of throwing a RemoteException.

METHOD-ID. "calculateCompoundInterest" THROWS "java.rmi.RemoteException".
DATA DIVISION.
LINKAGE SECTION.
  77 principle COMP-2.
  77 rate COMP-2.
  77 periods COMP-2.
  77 result COMP-2.
*
* The BY VALUE clause forces the Java-style usage of a variable;
* this means that COMP-2 will be a Java 'double' floating point
* in the definition.  A BY REFERENCE or BY CONTENT is suitable
* for communication with other Elastic COBOL programs, but for any
* interface or class which will have Java accessors, the BY VALUE
* is simpler.
*

PROCEDURE DIVISION USING BY VALUE principle, rate, periods GIVING result.
END-METHOD.

END-INTERFACE.
```

The remote interface specifies only one 'business method' called calculateCompoundInterest. This is the list of business methods available to any clients.

Home Interface

The home interface is even simpler. The home interface describes the creation method.

```
* This example is coded for the JBoss EJB Container.  The
* naming provider and setup information may differ for other
* EJB Containers.
* The logic flow remains the same. (http://www.jboss.org)
*
* This example requires J2EE.  JDK 1.2+ plus JBoss is sufficient.
*
* The CLASSPATH to compile must include \jboss\lib\ext\ejb.jar in
* addition to the ecobol.jar required for any COBOL program.
* This is the javax.ejb.* package; its location may vary in
* other EJB vendors.
*
* Set the package name; this could be done from the command line
* using the '-package name' directive instead.  Setting a package
* is generally not necessary for an EJB, but it's good practice.

$SET PACKAGE com.web_tomorrow.interest

*
* An INTERFACE-ID is the same as a CLASS-ID, but without any
* method definitions.  No special-names, no data other than
* linkage, and nothing in the procedure division other than
* the procedure division using is allowed.  This defines a
```

```
* contract which another class-id program may implement.
*

IDENTIFICATION DIVISION.
INTERFACE-ID. "InterestHome" INHERITS "javax.ejb.EJBHome".
*
* Only a create method is necessary to define for this interface,
* allowing the client to create the Enterprise JavaBean.
*
METHOD-ID. "create" THROWS "java.rmi.RemoteException", "javax.ejb.CreateException".
DATA DIVISION.
LINKAGE SECTION.
    77 result OBJECT REFERENCE "com.web_tomorrow.interest.Interest".
PROCEDURE DIVISION GIVING result.
END-METHOD.

END-INTERFACE.
```

EJB Class

Finally there is the Bean class. This is the only one that does any real work in this simple example.

```
* This example is coded for the JBoss EJB Container.  The naming
* provider and setup information may differ for other EJB
* Containers.
*
* The logic flow remains the same. (http://www.jboss.org)
*
* This example requires J2EE.  JDK 1.2+ plus JBoss is sufficient.
*
* The CLASSPATH to compile must include \jboss\lib\ext\ejb.jar in
* addition to the ecobol.jar required for any COBOL program.  This
* is the javax.ejb.* package; its location may vary in other EJB
* vendors.
*
* Set the package name; this could be done from the command line
* using the '-package name' directive instead.  Setting a package
* is generally not necessary for an EJB, but it's good practice.

$SET PACKAGE com.web_tomorrow.interest

*
* This class-id program is the main Enterprise JavaBean.
* It contains the business logic which is exposed to the outside
* world and enough hooks to allow the EJB Container to control
* the bean.
*
* A SessionBean is a particular type of EJB; this is the type most
* corresponding to a CICS transaction and the type most COBOL
* programs will implement.
*
* A SessionBean is a logic bean; an EntityBean is a data bean.

IDENTIFICATION DIVISION.
CLASS-ID. "InterestBean" IMPLEMENTS "javax.ejb.SessionBean".

* IDENTIFICATION DIVISION. is optional for a method-id.
```

```
METHOD-ID. "calculateCompoundInterest".

DATA DIVISION.
LINKAGE SECTION.
   01 principle COMP-2.
   01 rate      COMP-2.
   01 periods   COMP-2.
   01 result    COMP-2.

*
* This is the business method exposed to any EJB client.
*
* This example uses COMP-2, which is double precision floating
* point, rather than the more traditional COBOL types of fixed
* point arithmetic which would really be more appropriate.  The
* COMP-2 is used in this case so the signature of the method
* would exactly match the original Java program.  The BY VALUE
* forces normal Java types rather than COBOL types; this is
* the best practice for use in classes which may interact with
* other Java programs.  To keep exact datatypes, it's recommended
* to use PIC X(n) variables and move the values to the desired
* types; this translates to the Java String type which would
* both keep precision and is easy to use from Java as well as
* COBOL.
*

PROCEDURE DIVISION USING BY VALUE principle, rate, periods GIVING result.
MAIN-PARAGRAPH.

* The DISPLAY UPON SYSOUT goes to the main log file of the
* EJB Container. This may just be printed on the EJB Container
* sysout.  It will _not_ * be visible to the client.  Only the
* GIVING result piece will be returned and made visible to the
* client.

   DISPLAY "Someone called 'calculateCompoundInterest' in Elastic COBOL!"
      UPON SYSOUT
   DISPLAY " principle=" principle
      UPON SYSOUT
   DISPLAY " rate     =" rate
      UPON SYSOUT
   DISPLAY " periods  =" periods
      UPON SYSOUT

* There are other ways of computing this, but this example
* demonstrates the similarities between the Java and COBOL
* implementations of EJBs.

   COMPUTE result = principle * ((1+rate) ** periods) - principle

   DISPLAY " result   =" result
      UPON SYSOUT
   DISPLAY " formula is principle*((1+rate)**periods)-principle"
      UPON SYSOUT
   .

END-METHOD.

* Many EJB's can just copy the remaining code into their code.
*
* All remaining methods in this class are structural methods,
* necessary not for the business logic but rather for the EJB
* Container to be able to control this bean.
```

```
*
* Add in some DISPLAY UPON SYSOUT's to the procedure division
* of the methods in order to gain some feel over when these
* methods are called.
*
* We are given the session context, but we don't need it in this
* program so we ignore it.  This method is required to fulfill
* the interface of javax.ejb.SessionBean.
*

METHOD-ID. "setSessionContext".
DATA DIVISION.
LINKAGE SECTION.
   01 sc OBJECT REFERENCE "javax.ejb.SessionContext".

PROCEDURE DIVISION USING BY VALUE sc.
END-METHOD.

* The following methods are required to fulfill the interface
* of javax.ejb.SessionBean.  We don't need to do anything special,
* though, so we just create the method without a body.  When
* the method has no parameters and no result, this is the barest
* possible method definition.

METHOD-ID. "ejbCreate". END-METHOD.
METHOD-ID. "ejbRemove". END-METHOD.
METHOD-ID. "ejbActivate". END-METHOD.
METHOD-ID. "ejbPassivate". END-METHOD.

END-CLASS.
```

Notice that most of the methods are empty; they have to exist because they're specified by the SessionBean interface, but they don't need to do anything in this case.

This example uses COMP-2 for data storage. This is really an inappropriate type because floating point is inherently inexact for a currency usage such as this example, but this code is done to parallel the original Java code which used the equivalent data type. This allows the Java client to call the Elastic COBOL server, or the Elastic COBOL client to call the Java server.

All the PROCEDURE DIVISION USING and INVOKING USING for EJB Components are done BY VALUE rather than BY REFERENCE or BY CONTENT. This is important to ensure that interoperability with Java is preserved and that the parameters may be passed correctly across the network. The interfaces generated with INTERFACE-ID are Java source; these Java source files are readable by a Java programmer and are sufficient to allow a Java programmer to access the functionality.

If you haven't already done so, you should create and compile these .cob files in the directory \ecobol\samples\ejb\server.

Deployment Descriptor

With the .class files ready, it's time to create the deployment descriptor. The deployment descriptor is identical for Java or COBOL. In current versions of the EJB specification, it is an XML description of the EJB Component, its

bean and interfaces. Follow the directions provided by the EJB Container vendor for this information.

The .xml file will be placed in the META-INF subdirectory of the EJB Component. In this case, it's \ecobol\samples\ejb\server\META-INF\ejb-jar.xml.

```
<?xml version="1.0" encoding="Cp1252"?>

<ejb-jar>
  <description>JBoss test application </description>
  <display-name>Test</display-name>
  <enterprise-beans>
    <session>
      <ejb-name>Interest</ejb-name>
      <home>com.web_tomorrow.interest.InterestHome</home>
      <remote>com.web_tomorrow.interest.Interest</remote>
      <ejb-class>com.web_tomorrow.interest.InterestBean</ejb-class>
      <session-type>Stateless</session-type>
      <transaction-type>Bean</transaction-type>
    </session>
  </enterprise-beans>
</ejb-jar>
```

JBoss can use some additional information in a jboss.xml file. In this case, it's \ecobol\samples\ejb\server\META-INF\jboss.xml.

```
<jboss>
  <secure>false</secure>
  <container-configurations />
  <resource-managers />
  <enterprise-beans>
    <session>
      <ejb-name>Interest</ejb-name>
      <jndi-name>interest/Interest</jndi-name>
      <configuration-name></configuration-name>
    </session>
  </enterprise-beans>
</jboss
```

Packaging and Deploying the EJB Component

EJB Client

An EJB Client can be any number of types of program. It can be an application, a server, a Servlet, an Applet, a JSP page, or another EJB Component. EJB Components communicate with each other as if they were clients; by loosely connecting components in this manner, the EJB Components are free to be fully controlled by the EJB Container.

The following is a sample test client.

```
* Enterprise JavaBean Client Sample
*
* for the JBOSS EJB Container.
*
* The original Java version of this EJB client is available at:
```

```
* http://www.jboss.org/documentation/jboss_win32_5.html
*
* This simple application tests the `Interest'
* Enterprise JavaBean which is
* implemented in the package `com.web_tomorrow.interest'.
* For this to work, the Bean must be deployed on an EJB server.
*
* IMPORTANT If you want to test this in a real client-server
* configuration, this class goes on the client;
* the URL of the naming provider For this to work, the
* be changed from `localhost:1099' to the URL of the
* naming service on the server.
*
* Note: In COBOL, this program may also be used as a Servlet
* if compiling with the -Servlet flag, or using the
* com.heirloomcomputing.ecs.run.servlet as the
* name of the Servlet with the Servlet parameter 'Servlet' pointing
* to interest_client.  The only differences for Servlets would be
* to add some additional DISPLAYs of HTML elements surrounding the
* text, such as
* DISPLAY "<HTML><HEAD><TITLE>EJB Client</TITLE></HEAD>"...
*
* The CLASSPATH must include ejb.jar (found in \jboss\lib\ext\ejb.jar
* and other J2EE implementations), the server-side classes (only
* the pieces actually referenced), and ecobol.jar (already setup)
* for compilation.

IDENTIFICATION DIVISION.
PROGRAM-ID. interest-client.

ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
REPOSITORY.

* All classes are defined in either the CONFIGURATION SECTION/
* ENVIRONMENT DIVISION/REPOSITORY, or in the DATA DIVISION/CLASS-CONTROL.
* The format of either is identical.
*
  CLASS InitialContext IS "javax.naming.InitialContext"
  CLASS InterestHome IS "com.web_tomorrow.interest.InterestHome"
  CLASS InterestClass IS "com.web_tomorrow.interest.Interest"
  CLASS PortableRemoteObject IS "javax.rmi.PortableRemoteObject"
  .

DATA DIVISION.

* In the Java original, the variable declarations are intermixed with
* the procedural code.
*

WORKING-STORAGE SECTION.

* Declare the variables actually used by the program.
* The COMPOUND-INTEREST was originally done in a different format
* for Java, but the INVOKE GIVING may reference a traditional COBOL
* variable, allowing a more appropriate
* display format to be chosen, such as the numeric-edited item below.

      77 compoundInterest PIC $$$$,$$$,$$$.99.

      77 jndiContext OBJECT REFERENCE InitialContext.
      77 home OBJECT REFERENCE InterestHome.
      77 ref OBJECT REFERENCE.
```

```
       77 interest OBJECT REFERENCE InterestClass.

PROCEDURE DIVISION.

* This method does all the work. It creates an instance of
* the Interest EJB on the EJB server, and calls its
* `calculateCompoundInterest' method, then
* prints the result of the calculation.

MAIN.
       SET ENVIRONMENT "java.naming.factory.initial"
          TO        "org.jnp.interfaces.NamingContextFactory"

* Set up the naming provider; this may not always be necessary,
* dependingon how your Java system is configured.

       SET ENVIRONMENT "java.naming.provider.url"
          TO        "localhost:1099"

* Java Note:
* Enclosing the whole process in a single `try' block is not an
* ideal way to do exception handling, but I don't want to clutter
* the program up with catch blocks

* COBOL Note:
* No TRY block is necessary for this.  Rather, each
* INVOKE is automatically safe, catching its own exceptions.
* If you want notification when an invoke fails, code the
* individual ON EXCEPTION clause for the INVOKE.

* Get a naming context
       INVOKE InitialContext GIVING jndiContext
       ON EXCEPTION
          DISPLAY "Could not create InitialContext."
          GOBACK
       END-INVOKE

       DISPLAY "Got context"

* Get a reference to the Interest Bean
       INVOKE jndiContext "lookup"
          USING BY VALUE "interest/Interest" GIVING ref
       ON EXCEPTION
          DISPLAY "Could not lookup interest/Interest"
          GOBACK
       END-INVOKE

* Note that if you did not use jboss.xml to overwrite JNDI naming
* the object will be available under "Interest" its ejb-name
* INVOKE jndiContext "lookup" USING BY VALUE "Interest" GIVING ref

       DISPLAY "Got reference"

* Get a reference from this to the Bean's Home interface

       INVOKE PortableRemoteObject "narrow"
          USING BY VALUE ref InterestHome GIVING home
       ON EXCEPTION
          DISPLAY "Could not narrow."
          GOBACK
       END-INVOKE

* Create an Interest object from the Home interface
```

```
       INVOKE home "create" GIVING interest
       ON EXCEPTION
          DISPLAY "Could not create home."
          GOBACK
       END-INVOKE

      * call the calculateCompoundInterest() method to do the calculation

       INVOKE interest "calculateCompoundInterest"
          USING BY VALUE 1000 0.10 2
          GIVING compoundInterest
       ON EXCEPTION
          DISPLAY "Could not calculateCompoundInterest"
          GOBACK
       END-INVOKE

       DISPLAY "Interest on 1000 units, at 10% per period, "
            & "compounded over 2 periods is:" compoundInterest
       .
```

The SET ENVIRONMENT statements set Java System Properties. Two in particular are used in access from a client; these set the class used as a naming provider and where the EJB Container is located. Combined, these settings give enough information to find the correct EJB Container implementation.

The lookup finds the individual application/bean on the designated EJB Container.

The narrowing and creation create a proxy object on the client which represents the equivalent object on the server. Only methods defined in the external interface are available; but all calls to this returned object are actually performed on the server. All calls to the object are done using the INVOKE corresponding to the external METHOD-ID in the remote interface.

In this example, every INVOKE has an ON EXCEPTION clause which handles errors. Remember, EJB access is done across the network; this means that EJB objects may fail at any call. The EJB Container will often try to transparently correct certain errors on the server side, but if the client's network cord is pulled in the middle of the call, there is no way it can completely successfully. It's best to always do error handling when the object is remote. (Certain EJB Containers can be setup to control multiple machines such that even machine failure may be circumvented.)

The INVOKE of the business method in this case gives its result to a numeric-edited item. The parameters to an INVOKE must be well-matched, but the GIVING acts as a MOVE of the actual data to the GIVING item. The double precision floating point may be moved to a numeric-edited item, so this operation is allowed.

Client Output

```
Got context
Got reference
Interest on 1000 units, at 10% per period, compounded over 2 periods is:        $210.00
```

Server Output:

```
[Interest] Someone called 'calculateCompoundInterest' in Elastic COBOL!
[Interest]  principle=1000.0
[Interest]  rate     =0.1
[Interest]  periods  =2.0
[Interest]  result   =210.0000000000002
[Interest]  formula is principle*((1+rate)**periods)-principle
```

Appendix A – ASCII Table

(Zero Page Unicode)

Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code
0	00	NUL	32	20	space	64	40	@	96	60	`
1	01	SOH	33	21	!	65	41	A	97	61	a
2	02	STX	34	22	“	66	42	B	98	62	b
3	03	ETX	35	23	#	67	43	C	99	63	c
4	04	EOT	36	24	$	68	44	D	100	64	d
5	05	ENQ	37	25	%	69	45	E	101	65	e
6	06	ACK	38	26	&	70	46	F	102	66	f
7	07	BEL	39	27	‘	71	47	G	103	67	g
8	08	BS	40	28	(	72	48	H	104	68	h
9	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Note: Values are based off of 0-255 scale. Some COBOL functions require a 1-256 scale; add 1 for offset.

Appendix B – EBCDIC Table

Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code
0	00	NUL	32	20		64	40	spac e	96	60	-
1	01	SOH	33	21		65	41		97	61	/
2	02	STX	34	22		66	42		98	62	
3	03	ETX	35	23		67	43		99	63	
4	04		36	24		68	44		100	64	
5	05	HT	37	25	LF	69	45		101	65	
6	06		38	26	ETB	70	46		102	66	
7	07	DEL	39	27	ESC	71	47		103	67	
8	08		40	28		72	48		104	68	
9	09		41	29		73	49		105	69	
10	0A		42	2A		74	4A	[	106	6A	|
11	0B	VT	43	2B		75	4B	.	107	6B	,
12	0C	FF	44	2C		76	4C	<	108	6C	%
13	0D	CR	45	2D	ENQ	77	4D	(	109	6D	_
14	0E	SO	46	2E	ACK	78	4E	+	110	6E	>
15	0F	SI	47	2F	BEL	79	4F	| !	111	6F	?
16	10	DLE	48	30		80	50	&	112	70	
17	11		49	31		81	51		113	71	
18	12		50	32	SYN	82	52		114	72	
19	13		51	33		83	53		115	73	
20	14		52	34		84	54		116	74	
21	15		53	35		85	55		117	75	
22	16	BS	54	36		86	56		118	76	
23	17		55	37	EOT	87	57		119	77	
24	18	CAN	56	38		88	58		120	78	
25	19	EM	57	39		89	59		121	79	‘
26	1A		58	3A		90	5A	!]	122	7A	:
27	1B		59	3B		91	5B	$	123	7B	#
28	1C	IFS	60	3C		92	5C	*	124	7C	@
29	1D	IGS	61	3D	NAK	93	5D	)	125	7D	‘
30	1E	IRS	62	3E		94	5E	;	126	7E	=
31	1F	IUS	63	3F	SUB	95	5F	^	127	7F	“

Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code	Dec	Hex	Code
128	80		160	A0		192	C0	{	224	E0	\
129	81	a	161	A1	~	193	C1	A	225	E1	
130	82	b	162	A2	s	194	C2	B	226	E2	S
131	83	c	163	A3	t	195	C3	C	227	E3	T
132	84	d	164	A4	u	196	C4	D	228	E4	U
133	85	e	165	A5	v	197	C5	E	229	E5	V
134	86	f	166	A6	w	198	C6	F	230	E6	W
135	87	g	167	A7	x	199	C7	G	231	E7	X
136	88	h	168	A8	y	200	C8	H	232	E8	Y
137	89	i	169	A9	z	201	C9	I	233	E9	Z
138	8A		170	AA		202	CA		234	EA	
139	8B		171	AB		203	CB		235	EB	
140	8C		172	AC		204	CC		236	EC	
141	8D		173	AD		205	CD		237	ED	
142	8E		174	AE		206	CE		238	EE	
143	8F		175	AF		207	CF		239	EF	
144	90		176	B0		208	D0	}	240	F0	0
145	91	j	177	B1		209	D1	J	241	F1	1
146	92	k	178	B2		210	D2	K	242	F2	2
147	93	l	179	B3		211	D3	L	243	F3	3

148	94	m	180	B4		212	D4	M	244	F4	4
149	95	n	181	B5		213	D5	N	245	F5	5
150	96	o	182	B6		214	D6	O	246	F6	6
151	97	p	183	B7		215	D7	P	247	F7	7
152	98	q	184	B8		216	D8	Q	248	F8	8
153	99	r	185	B9		217	D9	R	249	F9	9
154	9A		186	BA		218	DA		250	FA	
155	9B		187	BB		219	DB		251	FB	
156	9C		188	BC		220	DC		252	FC	
157	9D		189	BD		221	DD		253	FD	
158	9E		190	BE		222	DE		254	FE	
159	9F		191	BF		223	DF		255	FF	

Note: Values are based off of 0-255 scale. Some COBOL functions require a 1-256 scale; add 1 for offset.

Appendix C – SQL

SQLCODE default values

	Description
0	success
100	no data
<0	error

SQLSTATE default values

Class	Subclass	Description
00	000	success condition
01	000	warning condition
01	001	cursor operation conflict
01	002	disconnection error
01	003	null value eliminated in set function
01	004	string data, right truncation
01	005	insufficient item descriptor areas
01	006	privilege not revoked
01	007	privilege not granted
01	008	implicit zero-bit padding
01	009	search condition too long for information schema
01	00A	query expression too long for information schema
02	000	no data
07	000	dynamic SQL error
07	001	using clause does not match dynamic parameter specification
07	002	using clause does not match target specifications
07	003	cursor specification cannot be executed
07	004	using clause required for dynamic parameters
07	005	prepared statement not a cursor specification
07	006	restricted data type attribute violation
07	007	using clause required for result fields
07	008	invalid descriptor count
07	009	invalid descriptor index
08	000	general connection exception
08	001	SQL client unable to establish SQL connection
08	002	connection name in use
08	003	connection does not exist
08	004	SQL server rejected SQL connection
08	006	connection failure
08	007	transaction resolution unknown
0A	000	feature not supported
0A	001	multiple server transactions
21	000	cardinality violation
22	000	data exception
22	001	string data, right truncation
22	002	null value, no indicator parameter
22	003	numeric value out of range
22	005	error in assignment
22	007	invalid datetime format
22	008	datetime field overflow
22	011	substring error

22	012	division by zero
22	015	inteval field overflow
22	018	invalid charcter value for cast
22	019	invalid escape character
22	021	character not in repertoire
22	022	indicator overflow
22	023	invalid paramteter value
22	024	unterminated C string
22	025	invalid escape sequence
22	026	string data, length mismatch
22	027	trim error
23	000	integrity constraint violation
24	000	invalid cursor state
25	000	invalid transaction state
26	000	invalid SQL statement name
27	000	triggered data change violation
28	000	invalid authorization specification
2A	000	syntax error or access rule violation in direct SQL statement
2B	000	dependent privilege descriptors still exist
2C	000	invalid character set name
2D	000	invalid transaction termination
2E	000	invalid connection name
33	000	invalid SQL descriptor name
34	000	invalid cursor name
35	000	invalid condition number
37	000	syntax error or access rule violation in dynamic SQL statement
3C	000	ambiguous cursor name
3D	000	invalid catalog name
3F	000	schema name
40	000	transaction rollback
40	001	serialization failure
40	002	integrity constraint violation
40	003	statement completion unknown
42	000	syntax error or access rule violation
44	000	with check option violation
HZ	000	remote database access

File Status Codes

Code	Description
00	OK
02	OK, duplicate key
04	OK, length mismatch on read; partial in XML
05	OK, optional file missing on open; file not deleted for DELETE FILE
07	OK, non-reel device on open or close
10	EOF, end of file for read, or first read on missing optional file
11	XML SAX warning
14	EOF, too many digits on read for relative file; XML start element
15	XML end element
16	XML start prefix
17	XML end pref
18	XML error processing
20	XML SAX error
21	invalid key, invalid sequence; XML skipped entity
22	invalid key, write key duplicate
23	invalid key, random record missing
24	invalid key, write beyond boundary

Code	Description
30	permanent error, no further info; XML SAX fatal error
34	permanent error, boundary violation
35	permanent error, non-optional file missing on open
37	permanent error, open mode not supported
38	permanent error, open locked file
39	permanent error, open attribute mismatch
41	logic error, already open
42	logic error, already closed
43	logic error, rewrite not after read; delete not after read
44	logic error, boundary; write too big, rewrite not same size as prior
46	logic error, already in error
47	logic error, not input file for read; not input file for start
48	logic error, not output file for write
49	logic error, not input-output file for rewrite; not input-output file for delete
51	record error, locked by other run unit
52	record error, deadlock
53	record error, max locks for run unit
54	record error, max locks for file connector
55	record error, read already locked
61	sharing error, file sharing conflict
90	device file not seekable
91	operation canceled
92	incorrect version
93	locking error, file locked (x/open); network error
99	locking error, record locked (x/open)

Appendix D – Compiler Options

Compiler directives are passed on the **ecobol.exe** (or **ecobol** on Linux/UNIX) command line or in the Elastic COBOL IDE Eclipse project property *COBOL Compiler Settings*.

Option	Description/Option Value
-help	Help on Elastic COBOL Options
-help:old	Help on Elastic COBOL Deprecated Options
-help:acu	Help on Elastic COBOL Acu-Style Options
-h	Help on Elastic COBOL Simple-Style Options
-dt	Data (0=EC,2=Acu,4=MPE,5=MF,6=RM,7=IBMA,8=IBME)
-dt:ec	Datatype Compatible with Elastic COBOL (default)
-dt:mf	Datatype Compatible with Micro Focus
-dt:acu	Datatype Compatible with AcuCOBOL
-dt:mpe	Datatype Compatible with HP MPE/iX COBOL-II
-dt:rm	Datatype Compatible with Liant RM/COBOL 7
-dt:bin	BINARY is BINARY-REV
-dt:truncbin	BINARY byte truncation rather than PIC truncation
-dt:compbin	COMPUTATIONAL is BINARY
-dt:aix	Datatypes Compatible with IBM AIX
-dt:os2	Datatypes Compatible with IBM OS/2
-source:auto	Auto-detect source format (default)
-source:free	Free-form source code
-source:fixed	Fixed-form source code
-source:variable	Variable-form source code
-source:utf8	UTF-8 encoding
-source:copypath	Set COPY library path to parameter
-source:suppress	Suppress parameter from being reserved word
-source:ignore	Ignore parameter reserved word
-source:wordlist	Set reserved word suppress list to filename parameter
-source:ansikey	Use only ANSI keywords
-source:basednumeric	Based Literals are Numeric (Hex,Decimal,Octal,Binary)
-source:hp	Prefer HP semantics when in conflict.
-source:path	Add parameter to front of PATH
-source:tabsize	Set tab size in spaces
-source:pause	Pause after compilation
-source:dbcs	Activate SHIFT-IN/SHIFT-OUT DBCS 0x0e, 0x0f
-source:dbcsso	Activate parameter as DBCS SHIFT-OUT
-source:dbcssi	Activate parameter as DBCS SHIFT-IN

-source:obsx	Flag Syntax Obsolete in COBOL 2002 or X/Open
-source:archaic	Flag Syntax Archaic in COBOL 2002
-source:assignenv	ASSIGN TO name is environment, not variable
-source:assignvar	ASSIGN TO name is variable, not environment
-out:ecoboldir	path to directory containing ecobol.dir
-out:java	Compile COBOL to .java (default) v1.5
-out:javaversion	Compile COBOL to specified java version
-out:class	Compile COBOL to .java, then executable .class
-out:execute	Compile COBOL to .java, .class, then execute
-out:cobolinjava	Include COBOL statements in .java
-out:nocobolinjava	Do not include COBOL statements in .java
-out:manyfields	Each 01 level in own constructor in .java
-out:name	Use parameter as name for applet/application
-out:subprogram	Designate as subprogram
-out:package	Use parameter as package name
-out:html	Generate template HTML file
-out:nohtml	Do not generate template HTML file
-out:bean	Create CobolBean interface
-out:servlet	Create Servlet interface
-out:nobean	Do not create CobolBean interface
-out:nothreadlock	Disable ThreadQueue suspension
-out:dir	Set output directory to parameter
-out:japplet	Inherit from JApplet by default
-out:applet	Inherit from Applet by default
-out:suppress	Suppress generated output (check only)
-out:trace	Enable READY|RESET TRACE
-out:filter	Set output filter (0=native,1=EBC-to-ASC,2=ASC-to-EBC, 3=breakup large classes)
-out:noarrayprops	No Array GET|SET Properties
-out:assert	Enable ASSERT functionality
-out:staticinvoke	Statically bind OO invokes where possible
-out:notransient	Do not produce transient modifiers
-out:smap	Generate .smap file
-out:nopretty	Do not create pretty java code
-out:transaction	Allow EXEC TRANSACTION capabilities
-out:indexcheck	Check table indexes against table bounds
-out:noindexcheck	Do not check table indexes against table bounds
-out:indexsize	Set INDEX byte size to param
-out:cicsmsg	Set transaction message level
-out:transactionmsg	Set transaction message level
-out:linkage	DFHEIBLK and DFHCOMMAREA variables creation (NO or YES 'default')

-err:file	Set error filename to parameter
-err:none	No error output
-err:stdout	Send errors to stdout
-err:stderr	Send errors to stderr
-err:max	Set maximum reported errors to parameter
-err:level	Minimum error Level (0=All, 1=Warning, 2=Error)
-err:pedantic	Pedantic error messages
-script:execute	Execute scriptfile upon completion
-script:append	Append parameter to end of scriptfile
-script:format	Format each line of scriptfile according to parameter
-script:javac	Use parameter as Java Compiler string
-script:java	Use parameter as Java Runtime string
-run:trace	Produce tracing messages for PARAGRAPH/SECTION, CALL, FILE I/O operations & SQL I/O operations
-run:tracecall	Produce tracing messages for each CALL.
-run:traceio	Produce tracing messages for each FILE I/O operation.
-run:tracepara	Produce tracing messages for each PARAGRAPH/SECTION.
-run:tracesql	Produce tracing messages for each SQL I/O operation.
-run:notrace	Turn off all tracing messages.
-run:notracecall	Turn off tracing messages for each CALL.
-run:notraceio	Turn off tracing messages for each FILE I/O operation.
-run:notracepara	Turn off tracing messages for each PARAGRAPH/SECTION.
-run:notracesql	Turn off tracing messages for each SQL I/O operation.
-run:visiblecall	Display literal CALL's.
-run:defaultbyte	Fill initial memory with byte number or 'space'
-run:noprogressbar	Do not generate runtime progress bar during download
-run:nocheckversion	Do not generate runtime code to verify Java version
-run:performrecurse	Use MF style perform
-run:noperformrecurse	Use ANSI style perform
-run:novisiblecallfail	Disable Visible Call Failure
-run:visiblecallfail	Enable Visible Call Failure
-run:dynvisiblecallfail	Allow runtime to enable/disable Visible Call Failure
-run:novisibleopenfail	Disable Visible Open Failure
-run:visibleopenfail	Enable Visible Open Failure
-run:dynvisibleopenfail	Allow runtime to enable/disable Visible Open Failure
-run:redefinesinfo	Preserve REDEFINES info at runtime for Datatype
-file:shareallothers	Default share: SHARING ALL OTHER
-file:sharenoother	Default share: SHARING NO OTHER
-file:sharereadonly	Default share: SHARING READ ONLY
-file:sharenone	Default share: no default/explicit sharing

-file:$infilename	dollar symbol ($) may be part of file name
-cache:enable	Enable cache always
-cache:disable	Disable cache always
-cache:auto	Automatic cache control (default)
-sql:off	Disable SQL Support. Abends on SQL statements
-sql:len	Set SQL VARYING stub len to param (len)
-sql:txt	Set SQL VARYING stub txt to param (arr)
-sql:next	SQL JDBC/DB2 next() workaround
-sql:db2	SQL IBM JDBC/DB2 workarounds
-sql:cro	SQL CONCUR_READ_ONLY
-sql:ta	SQL Transaction Adjust
-sql:sc	SQL Server handles Concat (||)
-sql:it	SQL Ignore Right Truncation
-sql:jdbc	SQL JDBC level to parameter (1,2,3,4)
-sql:sp	Generate SQL Stored Procedure of type "param": db2 only supported at this time.
-sql:logmode	SQL Logging (NO, YES)
-sql:groupmode	SQL group items treated as single item
-sql:nogroupmode	SQL group items treated as multiple items
-sql:nowarn	SQL Warning not obtained from JDBC
-sql:declarestatic	SQL DECLARE CURSOR always static
-sql:declaredynamic	SQL DECLARE CURSOR dynamic when possible
-sql:opt	Pass SQL option, -sql:opt ? for dump
-sql:mode	SQL in given parameter mode (ANSI, ORACLE, DB2)
-sql:cc	SQL Close on Commit to parameter (YES, NO)
-sql:eof	SQL END OF FETCH value (100,1403)
-sql:unsafenull	SQL Unsafe Null (NO,YES)
-sql:picx	SQL PICX type (CHARF,VARCHAR2)
-sql:url	Check SQL syntax against database URL (e.g., jdbc:postgresql://localhost/mydb)
-sql:user	Check SQL syntax against database USER (e.g., postgres)
-sql:password	Check SQL syntax against database PASSWORD (e.g., mypw
-sql:driver	Check SQL syntax against database DRIVER (e.g., com.postgresql.Driver)
-listing:xml	Listing file XML form to programid.xml
-listing:body	Listing file XML includes source body
-listing:define	Listing file XML includes 'define' tag
-listing:file	Listing file set to mainfile.list
-listing:cross	Listing file includes cross-reference
-listing:vbref	Listing file cross-reference includes verbs
-listing:info	Listing file cross-reference includes info lines

-listing:all	Listing file cross-reference includes all options
-listing:dir	Listing files output directory
-cp	Use parameter as classpath

Appendix E – Runtime Options

Elastic COBOL runtime options are applied as Java defined runtime options,

```
java –Dname=value –jar myproj.jar
```

as parameters to the application,

```
java–jar myproj.jar name=value
```

or included as part of an Elastic COBOL configuration file in the current working directory or **/etc** directory named **cblconfig**, **cblconfi**, **cobopt.cfg** or other file as set with the **CBLCONFIG** property set with `–DCBLCONFIG=/tmp/my-ec.conf`

```
name=value
```

Option	Option Value	Description
DDNAME	DATASETNAME	When **-source:assignenv** compiler option is specified, associate DDNAMEs from HCI Batch Platform (JES/JCL) referenced in COBOL programs to dataset (file) names on disk
DDNAME.DISP	SHR, OLD, MOD	When **-source:assignenv** compiler option is specified, set the disposition of the COBOL connection for the SELECT statement. By default, INPUT is shared (SHR) disposition and OUTPUT and I O are exclusive (OLD) disposition. Applications running in HCI Batch Platform (JES/JCL) with DISP=MOD set the equivalent of EXTEND mode although the program opens for OUTPUT.
FILESYSTEM	ECOBOL, MICROFOCUS, ACUCOBOL, ACUCONNECT, ISAM, OS400	Changes the default filesystem used by Elastic COBOL. If a protocol is not specified in an ASSIGN TO, the default filesystem is used where applicable.
FILESYSTEMSEQ	ECOBOL, MICROFOCUS, ACUCOBOL, ACUCONNECT, ISAM, OS400	Changes the default filesystem for sequential files. If a protocol is not specified in an ASSIGN TO, the default filesystem is used where applicable.
FILESYSTEMREL	ECOBOL, MICROFOCUS, ACUCOBOL, ACUCONNECT, ISAM, OS400	Changes the default filesystem for relative files. If a protocol is not specified in an ASSIGN TO, the default filesystem is used where applicable.
FILESYSTEMIDX	ECOBOL, MICROFOCUS, ACUCOBOL, ACUCONNECT,	Changes the default filesystem for indexed files. If a protocol is not specified in an ASSIGN TO, the default filesystem is used where applicable.

	ISAM, OS400	
IDXCACHEMODE	READONLY, READWRITE	Changes the default indexed file cache mode setting. Read only caching is slower, but also safer. NOTE: If this value is not set and the indexed file is open for MASS UPDATE, then read-write caching is used.
IDXCACHE	default	Changes the indexed file cache size percentage setting. To double the cache size use 200.0, to halve it use 50.0. This setting enables fine tuning of indexed file performance in some cases.
CONSOLEFG	BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, BROWN, WHITE, BRIGHT-BLACK, BRIGHT-BLUE, BRIGHT-GREEN, BRIGHT-CYAN, BRIGHT-RED, BRIGHT-MAGENTA, BRIGHT-YELLOW, BRIGHT-BROWN, BRIGHT-WHITE	Foreground color used by default in the graphically emulated text console.
CONSOLEBG	BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, YELLOW, BROWN, WHITE, BRIGHT-BLACK, BRIGHT-BLUE, BRIGHT-GREEN, BRIGHT-CYAN, BRIGHT-RED, BRIGHT-MAGENTA, BRIGHT-YELLOW, BRIGHT-BROWN, BRIGHT-WHITE	Background color used by default in the graphically emulated text console.
SECURE-CHAR	*	This character is used in password fields as a substitute for sensitive information.
LINES	25	Changes the default number of screen lines for the console.
COLUMNS	80	Changes the default number of screen columns for the console.
CURSOR-MODE	1, 2, 3	Determines when the cursor should be visible: 1=Always, 2=Never, and 3=Only during ACCEPT.
CONSOLE-FONT	default	Override the default console font where applicable.
CONSOLE-FONT-	default	Override the default console font size where

SIZE		applicable.
CONSOLE-CELL	default	Overrides the console's font width setting.
CONSOLE-WIDTH-MAX	default	Sets the maximum width of font allowed.
CONSOLE-WIDTH-MIN	default	Sets the minimum width of font allowed.
CONSOLE-WIDTH-MULT	default	Sets a multiplier for the font width heuristic.
CONSOLE	default	List of console font settings, separated by commas, in this order: CONSOLE-CELL, CONSOLE-WIDTH, CONSOLE-WIDTH-MIN, CONSOLE-WIDTH-MAX, CONSOLE-WIDTH-MULT, CONSOLE-FONT.
USER-GRAY	default	Used to override color number 7, low intensity white. This value should be a decimal, octal, or hexadecimal 24-bit integer (RGB).
USER-WHITE	default	Used to override color number 15, high intensity white. This value should be a decimal, octal, or hexadecimal 24-bit integer (RGB).
PFTERMS	default	Controls function key terminator assignments. For example: 4-6,!9,10 will assign F4,F5,F6 and F10 to be user defined function key terminators. F9 will be disabled as a terminator, and the function keys that are not in the list will be assigned as system defined function key terminators.
TERMINATE	ENTER, ESCAPE, CONTROL-x	Controls normal terminator assignments. For example: ESCAPE, ENTER, CONTROL-G will allow the escape, enter, and control-G keys to terminate an ACCEPT.
KEYCODESTYLE	ACU, MF, CHAR, JAVA	Controls which raw key codes are returned in CRT-STATUS. For example, JAVA will return VK_ KeyEvent codes, CHAR will return ASCII codes, MF returns Micro Focus style codes, ACU returns AcuCOBOL style codes.
MQSERVER	localhost	Assigns MQSeries server hostname or IP address.
MQ_HOSTNAME	localhost	Assigns MQSeries client hostname or IP address.
MQ_PORT	default	Assigns MQSeries port number.
MQ_CHANNEL	default	Assigns MQSeries channel setting.
MQ_USER_ID	none	Assigns MQSeries user identification.
MQ_PASSWORD	none	Assigns MQSeries user password.
CODE-SUFFIX	none	Appends a string to the end of program names for all CALL statements.
CODE-CASE	0, 1, 2	Forces uppercase or lowercase program names for all CALL statements. 0 = No force (default), 1 = Force lowercase, 2 = Force uppercase.
CODE-MAPPING	True, False	Enables program name mapping for CALL statements. At the time of a CALL, if CODE-

		MAPPING is enabled and a configuration parameter matching the CALL statement's program name is set to a value, the value of the parameter is used in place of the program name.
S1,S2,S3,S4, S5,S6,S7,S8, S9,S10,S11, S12,S13,S14, S15,S16,S17, S18,S19,S20, S21,S22,S23, S24,S25,S26	True, False	COBOL Switch values
DEBUGMODE	True, False	Execute in debugger if debug information present. This debug information must have been compiled into the program by setting the debug flag before compilation.
SHOWNUMERICERROR	True, False	Certain numeric errors may be detected at runtime that have an automatically corrected default behavior, bringing the numeric to zero. This condition may be made visible in certain cases by setting this flag.
DEFSYS	True, False	Default to SYSIN/SYSOUT rather than CONSOLE or other runtime default. This may also be specified at compilation time.
LOG_SQL	True, False	Enable verbose logging of runtime SQL activity; this is useful for tracking down lower-level activities of the SQL/JDBC access.
LOG	True, False	Enable general runtime logging.
file.encoding	Cp1252	Encoding for all I/O (file and display statements).
ibm.encoding	Cp1047	File encoding for IBM EBCDIC files when -dt:ibm compile time option has been specified
CHECKINDEX	True/1/on, False/0/off	Used along with compiler option -out:indexcheck. Its value determines what happens when an index out of bounds situation is detected. When set to true, it causes the runtime to throw an IndexOutofBounds CobolException and terminate the program. When set to false(default), it causes the runtime to return a ArrayIndexOutOfBoundsVariable object as the referenced item.
GARBAGE	DEFAULT, VERBOSE, QUIET	Used to determine the actions associated with the creation and interaction with a ArrayIndexOutOfBoundsVariable object. Default behavior is to return 0/null when accessed and throw a CobolException if an attempt is made to move a value to it. VERBOSE is similar to DEFAULT but also logs a message if Logging is set to true. The

		messages get logged when a IndexOutofBounds error occurs and also when the ArrayIndexOutOfBoundsVariable throws an exception. QUIET is same as DEFAULT but does not throw an exception when a value is moved to the object.
RUN_TRACE	1 (on), 0 (off)	If application has been compiled with -run:trace[para|call|io|sql], then all trace information can be suppressed by setting to 0.
RUN_TRACEPARA	1 (on), 0 (off)	If application has been compiled with -run:trace or -run:tracepara, then all PARAGRAPH trace information can be suppressed by setting to 0. If RUN_TRACE=0, then this setting is ignored.
RUN_TRACECALL	1 (on), 0 (off)	If application has been compiled with -run:trace or -run:tracecall, then all CALL trace information can be suppressed by setting to 0. If RUN_TRACE=0, then this setting is ignored.
RUN_TRACEIO	1 (on), 0 (off)	If application has been compiled with -run:trace or -run:traceio, then all FILE I/O trace information can be suppressed by setting to 0. If RUN_TRACE=0, then this setting is ignored.
RUN_TRACESQL	1 (on), 0 (off)	If application has been compiled with -run:trace or -run:tracesql, then all SQL I/O trace information can be suppressed by setting to 0. If RUN_TRACE=0, then this setting is ignored.

IndexA