isCOBOL Evolve: EIS

Extend Internet System

Key Topics:

- The Service Bridge facility
- isCOBOL and AngularJS
- COBOL Servlet option (OOP)
- webClient option
- Web Direct 2.0 option
- HTTPHandler class (com.iscobol.rts.HTTPHandler)
- HTTPClient class (com.iscobol.rts.HTTPClient)

Table of Contents

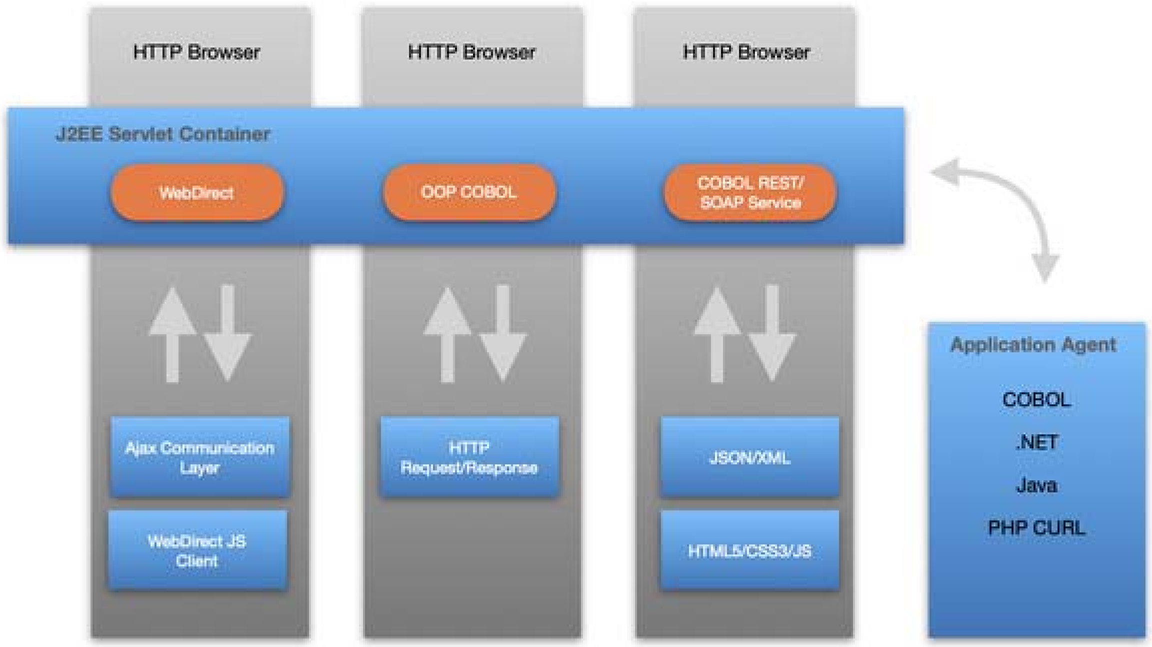
Chapter 1

Introduction

Overview

isCOBOL Extend Internet System (EIS) is an umbrella of tools and features available in the isCOBOL Evolve Suite that allows development and execution of a web based application in a J2EE container. There are several options to deploy a web application based on EIS as shown in the figure below, isCOBOL Extend Internet System Architecture, in order to provide the right option for every scenario.

isCOBOL Extend Internet System Architecture



Getting Started

The setup of isCOBOL EIS environment requires the following steps:

1. Download and install the Java Development Kit (JDK)

2. Download and install isCOBOL Evolve SDK
3. Activate the License

In order to activate your isCOBOL Evolve products, you will need the e-mail you received from Veryant containing your license key. Contact your Veryant representative for details.

Download and install the Java Development Kit (JDK)

A JDK must be installed on your machine in order to use isCOBOL EIS. For best results and performance, install the latest JDK version available for your platform. isCOBOL is certified to work correctly with both Oracle JDK and OpenJDK from version 8 to version 11.

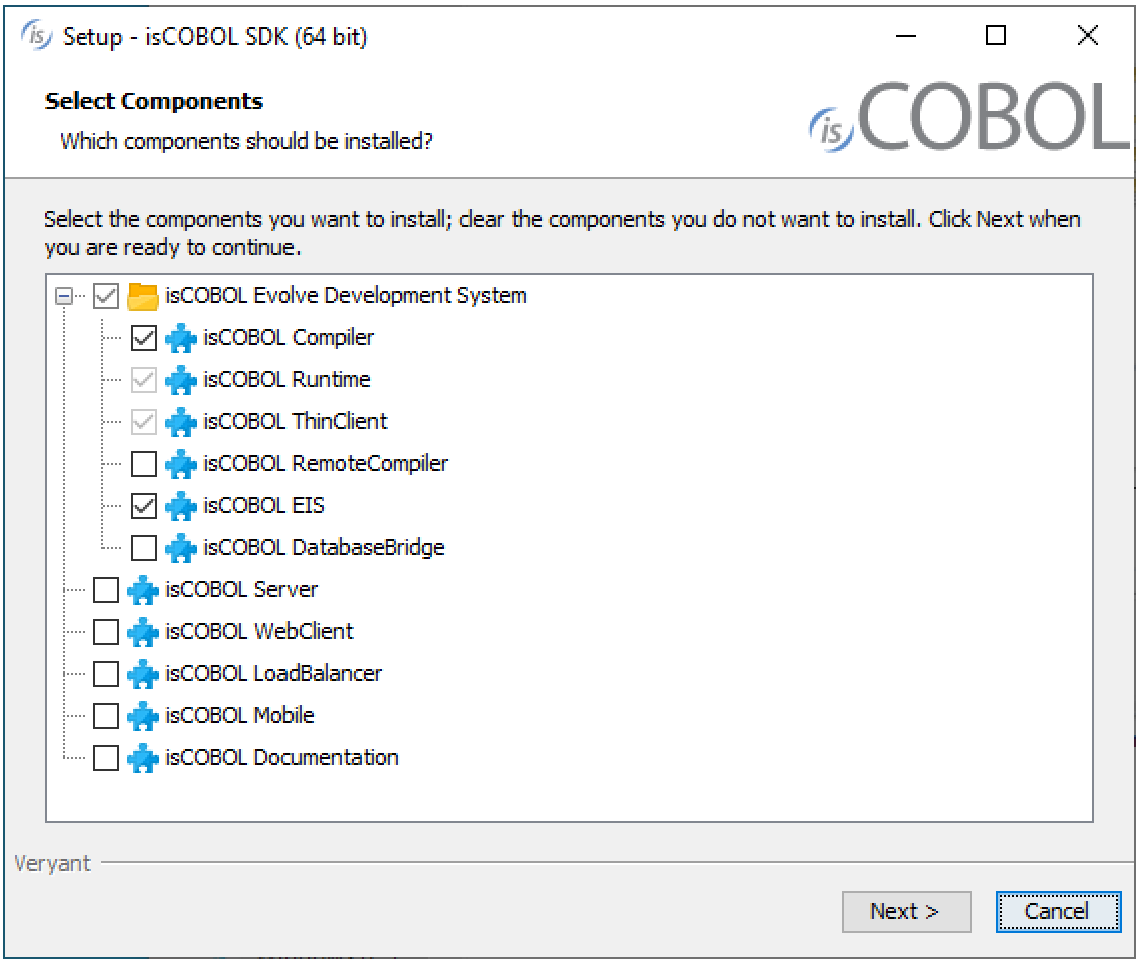
Self-extracting setups are provided for the Windows platform.

On Unix/Linux platforms Java may be already installed. If it’s not the case, you can install it using the appropriate system commands (e.g. yum, or apt-get).

Download and install isCOBOL Evolve SDK

Windows

1. If you haven't already done so, Download and install the Java Development Kit (JDK) .
2. Go to "https://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down to the list of files for Windows x64 64-bit or Windows x86 32-bit. Select isCOBOL_2021_R1_*n*_Windows.*arc*.msi, where *n* is the build number and *arc* is the system architecture.
6. Run the downloaded installer to install the files.
7. Select "isCOBOL EIS" from the list of products when prompted.


Setup - isCOBOL SDK (64 bit)

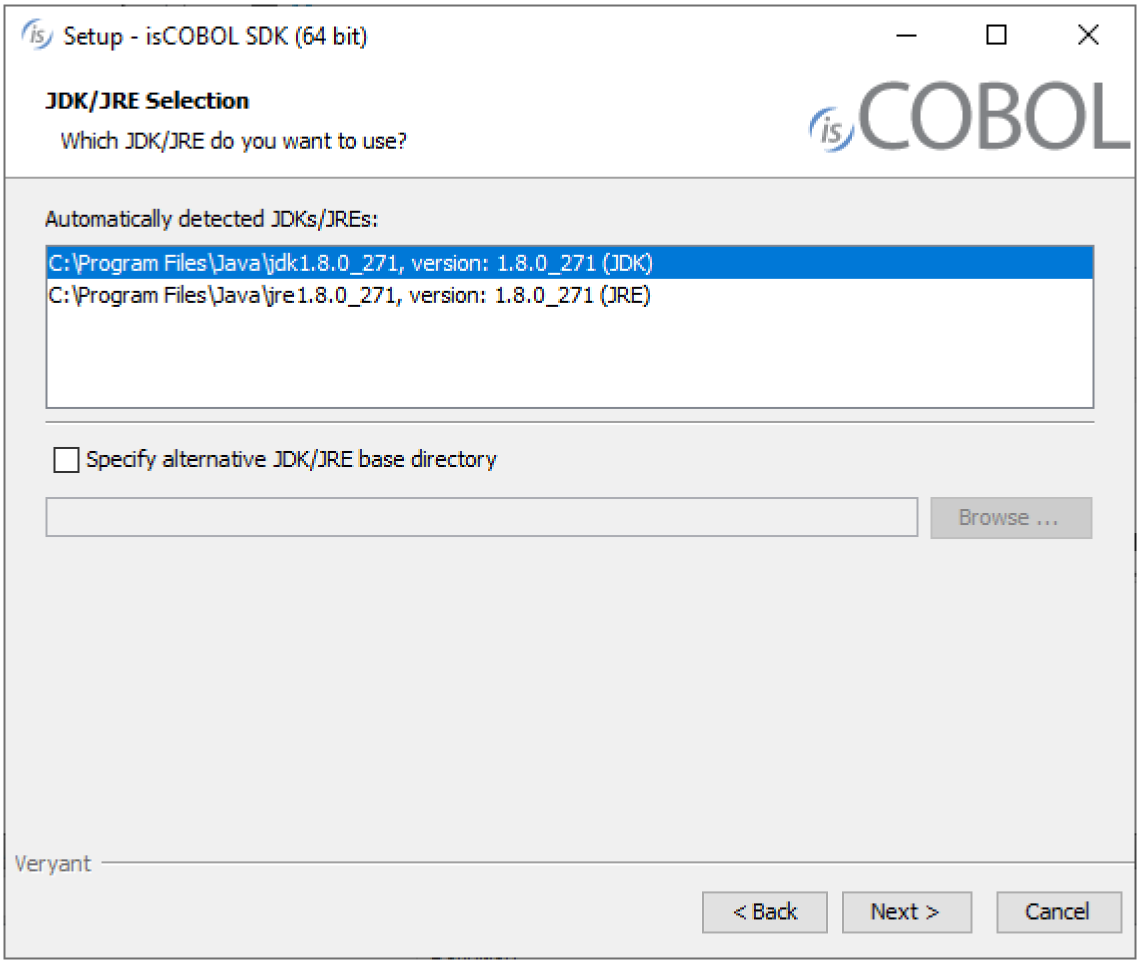
Select Components

Which components should be installed?

Select the components you want to install; clear the components you do not want to install. Click Next when you are ready to continue.

- [✓] isCOBOL Evolve Development System
  - [✓] isCOBOL Compiler
  - [✓] isCOBOL Runtime
  - [✓] isCOBOL ThinClient
  - [ ] isCOBOL RemoteCompiler
  - [✓] isCOBOL EIS
  - [ ] isCOBOL DatabaseBridge
- [ ] isCOBOL Server
- [ ] isCOBOL WebClient
- [ ] isCOBOL LoadBalancer
- [ ] isCOBOL Mobile
- [ ] isCOBOL Documentation

8. Select your JDK when prompted


Setup - isCOBOL SDK (64 bit)

JDK/JRE Selection

Which JDK/JRE do you want to use?

Automatically detected JDKs/JREs:

C:\Program Files\Java\jdk1.8.0_271, version: 1.8.0_271 (JDK)

C:\Program Files\Java\jre1.8.0_271, version: 1.8.0_271 (JRE)

☐ Specify alternative JDK/JRE base directory

9. Follow the wizard procedure to the end. In the process you will be asked to provide the installation path ("C:\Veryant" by default) and license keys. You can skip license activation and perform it later, as explained in Activate the License.

Linux, FreeBSD, Mac OSX and SunOS

1. If you haven't already done so, Download and install the Java Development Kit (JDK) .
2. Go to "http://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down, and select the appropriate .tar.gz file for the product and platform you require.
6. Extract all contents of the archive. For example,
   on Linux 32 bit:

```
gunzip isCOBOL_2021_R1_*_Linux.32.i586.tar.gz
tar -xvf isCOBOL_2021_R1_*_Linux.32.i586.tar
```

on Linux 64 bit:

```
gunzip isCOBOL_2021_R1_*_Linux.64.x86_64.tar.gz
tar -xvf isCOBOL_2021_R1_*_Linux.64.x86_64.tar
```

on FreeBSD:

```
gunzip isCOBOL_2021_R1_*_FreeBSD.64.tar.gz
tar -xvf isCOBOL_2021_R1_*_FreeBSD.64.tar
```

on Mac OSX:

```
gunzip isCOBOL_2021_R1_*_MacOSX.64.x86_64.tar.gz
tar -xvf isCOBOL_2021_R1_*_MacOSX.64.x86_64.tar
```

on SunOS:

```
gunzip isCOBOL_2021_R1_*_SunOS.64.tar.gz
tar -xvf isCOBOL_2021_R1_*_SunOS.64.tar
```

7. Change to the "isCOBOL2021R1" folder and run "./setup", you will obtain the following output:

```
===============================================================================

                           isCOBOL EVOLVE Installation
                           For isCOBOL Release 2021R1
                        Copyright (c) 2005 - 2021 Veryant

===============================================================================

Install Components:

    [0] All products...................................... (no)
    [1] isCOBOL Compiler (includes [2] & [3])............. (yes)
    [2] isCOBOL Runtime (includes [3]).................... (no)
    [3] isCOBOL ThinClient................................ (no)
    [4] isCOBOL RemoteCompiler............................ (no)
    [5] isCOBOL EIS....................................... (no)
    [6] isCOBOL DatabaseBridge............................ (no)
    [7] isCOBOL Server.................................... (no)
    [8] isCOBOL WebClient................................. (no)
    [9] isCOBOL LoadBalancer.............................. (no)
   [10] isCOBOL Mobile.................................... (no)

Install Path:
    [P] isCOBOL parent directory: UserHome

JDK Path:
    [J] JDK install directory: JavaHome

[S] Start Install        [Q] Quit

==============================================================================
Please press [ 1 2 3 4 5 6 7 8 P J S Q ]
```

8. Type "5", then press Enter to select isCOBOL EIS.
9. (optional) Type "P", then press Enter to provide a custom installation path, if you don't want to keep the default one.
10. Type "S", then press Enter to start the installation.

    Note - if the setup script is not available for your Unix platform or you don't want to use it, just extract the tgz content to the folder where you want isCOBOL to be installed.

isCOBOL Evolve for UNIX/Linux provides shell scripts in the isCOBOL "bin" directory for compiling, running, and debugging programs. These scripts make use of two environment variables, ISCOBOL to locate the isCOBOL installation directory and ISCOBOL_JDK_ROOT to locate the JDK installation directory. To use these scripts set these environment variables and add the isCOBOL "bin" directory to your PATH.

For example, if you install isCOBOL in "/opt/isCOBOL" and your JDK is in "/opt/java/jdk1.8.0":

```
export ISCOBOL=/opt/isCOBOL
export ISCOBOL_JDK_ROOT=/opt/java/jdk1.8.0
export PATH=$ISCOBOL/bin:$PATH
```

Other Unix

A dedicated setup is provided for the following Unix platforms:

- Linux 32 bit
- Linux 64 bit
- FreeBSD
- Mac OSX 64 bit
- SunOS

f you need to install isCOBOL on another Unix platform, you can use the platform independent setup.

This setup includes only the cross platform items while it lacks native items. Contact Veryant if you need the porting of a native item to your Unix platform.

Instructions for the installation of the platform independent setup are provided below.

1. If you haven't already done so, Download and install the Java Development Kit (JDK).
2. Go to "https://www.veryant.com/support".
3. Sign in with your User ID and Password.
4. Click on the "Download Software" link.
5. Scroll down to the "Platform Independent" section and select isCOBOL_2021_R1_*n*_noarch.tar.gz, where *n* is the build number.

Extract all contents of the archive:

```
gunzip isCOBOL_2021_R1_*_noarch.tar.gz
tar -xvf isCOBOL_2021_R1_*_noarch.tar
```

Distribution Files

For information on a specific distribution file, please see the README file installed with the product.

Activate the License

If you provided license keys during the installation, on Windows, you should skip reading this chapter.

isCOBOL EIS looks for the following configuration properties for the license keys at compile-time:

```
iscobol.compiler.license.2021=<license_key>
iscobol.eis.license.2021=<license_key>
```

isCOBOL EIS looks for the following configuration property for the license key at run-time:

```
iscobol.license.2021=<license_key>
```

These keys should be stored in one of the following files (if they exist):

Windows

1. \etc\iscobol.properties in the drive where the working directory is
2. C:\Users\<username>\iscobol.properties (the setup wizard saves licenses here, if you don’t skip activation)
3. iscobol.properties found in the Java Classpath
4. %ISCOBOL%\iscobol.properties
5. a custom configuration file passed on the command line

Unix/Linux

1. /etc/iscobol.properties
2. $HOME/iscobol.properties
3. iscobol.properties found in the Java Classpath
4. $ISCOBOL/iscobol.properties
5. a custom configuration file passed on the command line

NOTE - Files are listed in the order they’re processed. If the license key appears in more than one of the above files, then the last occurrence is considered.

Chapter 2

Web Service option

Introduction

A Web Service is a software system designed to support interoperable machine-to-machine interaction over a network.

Many organizations use multiple software systems for management. Different software systems often need to exchange data with each other, and a web service is a method of communication that allows two software systems to exchange this data over the internet. The software system that requests data is called a service requester or consumer, whereas the software system that would process the request and provide the data is called a service provider or producer.

Different software might be built using different programming languages, and hence there is a need for a method of data exchange that doesn't depend upon a particular programming language. Most types of software can, however, interpret *XML* or *JSON* tags. Thus, web services can use *XML* or *JSON* files for data exchange.

Two predominant web services frameworks, *REST* and *SOAP*, are used in web site development.

REST (Representational State Transfer) and *SOAP* (Simple Object Access Protocol) provide mechanisms for requesting information from resources, *REST*, or from endpoints, *SOAP*. Perhaps the best way to think of these technologies is as a method of making a remote procedure call against a well-defined API. *SOAP* has a more formal definition mechanism called WSDL, Web Services Definition Language, and is more complex to implement. *REST* uses the standard *HTTP* request and response mechanism, simplifying implementation and providing for a more flexible, loose coupling of the client and server. Note that *REST* also supports the transfer of non-XML messages such as *JSON* (JavaScript Object Notation).

The Service Bridge facility

The creation of SOAP and REST Web Services is made easy by the isCOBOL Server Bridge facility. With this feature enabled, every time the Compiler compiles a legacy COBOL program with Linkage Section, it generates a bridge class that allows the program to be used as a Web Service.

The feature is activated and controlled through the Library Routines Configuration entries that can be set either in the configuration file or directly in the source through the SET Directive. isCOBOL IDE users instead can rely on the isCOBOL Service Editor.

Let’s consider the legacy COBOL program "SONGS.cbl" installed among isCOBOL samples. This program allows the user to manage an archive of songs. It has the following Linkage Section:

```
       01  lnk-op-code              pic x.
           88 lnk-insert            value "I".
           88 lnk-update            value "U".
           88 lnk-read              value "R".
           88 lnk-read-next         value "N".
           88 lnk-read-previous     value "P".
           88 lnk-delete            value "D".
           88 lnk-first             value "F".
           88 lnk-last              value "L".
       01  lnk-song-data.
           05 lnk-sd-id             pic 9(5).
           05 lnk-sd-title          pic x(30).
           05 lnk-sd-length         pic x(5).
           05 lnk-sd-artist         pic x(20).
           05 lnk-sd-album          pic x(30).
           05 lnk-sd-genre          pic x(15).
           05 lnk-sd-label          pic x(30).
           05 lnk-sd-year           pic 9(4).
           05 lnk-sd-authors        occurs 5.
              10 lnk-sd-author      pic x(20).
       01  lnk-return-status.
           05 lnk-status            pic x(2).
              88 lnk-ok             value "OK".
              88 lnk-ko             value "KO".
           05 lnk-file-status       pic x(2).
           05 lnk-status-message    pic x(50).
```

In the next two chapters we’ll see how to transform it in a Web Service through the Service Bridge facility, with and without isCOBOL IDE.

Web Service generation at command-line

The only action required in the source code is the mapping between Linkage Section data items and Web Service parameters. If no action is taken, the Compiler generates a Web Service parameter for each elementary COBOL data item and the parameter is assumed to be input/output. In this case instead we wish to define the first group item as input, the second as i/o, and the third as output, because the archive record buffer is shared between caller and callee, while the other two parameters are one-way.

This is achieved through ELK Directives. The original Linkage Section code will change from

```
       01  lnk-op-code              pic x.
           88 lnk-insert            value "I".
           88 lnk-update            value "U".
           88 lnk-read              value "R".
           88 lnk-read-next         value "N".
           88 lnk-read-previous     value "P".
           88 lnk-delete            value "D".
           88 lnk-first             value "F".
           88 lnk-last              value "L".
       01  lnk-song-data.
           05 lnk-sd-id             pic 9(5).
           05 lnk-sd-title          pic x(30).
           05 lnk-sd-length         pic x(5).
           05 lnk-sd-artist         pic x(20).
           05 lnk-sd-album          pic x(30).
           05 lnk-sd-genre          pic x(15).
           05 lnk-sd-label          pic x(30).
           05 lnk-sd-year           pic 9(4).
           05 lnk-sd-authors        occurs 5.
              10 lnk-sd-author      pic x(20).
       01  lnk-return-status.
           05 lnk-status            pic x(2).
              88 lnk-ok             value "OK".
              88 lnk-ko             value "KO".
           05 lnk-file-status       pic x(2).
           05 lnk-status-message    pic x(50).
```

to

```
      $elk input
       01  lnk-op-code              pic x.
           88 lnk-insert            value "I".
           88 lnk-update            value "U".
           88 lnk-read              value "R".
           88 lnk-read-next         value "N".
           88 lnk-read-previous     value "P".
           88 lnk-delete            value "D".
           88 lnk-first             value "F".
           88 lnk-last              value "L".
       01  lnk-song-data.
           05 lnk-sd-id             pic 9(5).
           05 lnk-sd-title          pic x(30).
           05 lnk-sd-length         pic x(5).
           05 lnk-sd-artist         pic x(20).
           05 lnk-sd-album          pic x(30).
           05 lnk-sd-genre          pic x(15).
           05 lnk-sd-label          pic x(30).
           05 lnk-sd-year           pic 9(4).
           05 lnk-sd-authors        occurs 5.
              10 lnk-sd-author      pic x(20).
      $elk output
       01  lnk-return-status.
           05 lnk-status            pic x(2).
              88 lnk-ok             value "OK".
              88 lnk-ko             value "KO".
           05 lnk-file-status       pic x(2).
           05 lnk-status-message    pic x(50).
```

After this quick modification, we can compile the program as follows:

```
iscc -c=compiler.properties SONGS.cbl
```

The file compiler.properties should include one or more Library Routines Configuration entries. For the moment, we just activate the Sevice Bridge feature. It will generate a SOAP RPC Web Service. So, compiler.properties contains:

```
iscobol.compiler.servicebridge=1
```

At the end of the compilation process, you will find a folder named "servicebridge" with a source file inside:

- *soapSONGS.cbl* : the bridge program that allows our program to be called as Web Service

In the compiler output folder (the working directory, in this case) instead you will find:

- *SONGS.class* : the class of the COBOL program just compiled
- *SONGS.wsdl* : the XML descriptor of the service

Refer to the installed sample README file for instructions about the deployment and testing of these items.

We've just demonstrated how to create a SOAP Web Service. With the next step, we're going to generate a REST Web Service instead. To achieve it, change compiler.properties as follows:

```
iscobol.compiler.servicebridge=1
iscobol.compiler.servicebridge.type=REST
```

Then compile again with the command:

```
iscc -c=compiler.properties SONGS.cbl
```

At the end of the compilation process, you will find a folder named "servicebridge" with one file inside:

- *restSONGS.cbl* : the bridge program that allows our songs program to be called as REST Web Service.

Refer to the installed sample README file for instructions about the deployment and testing of these items.

An alternative way to activate the Service Bridge facitlity is to set the necessary properties directly in the source code, using the SET Directive. For example:

```
      $set "servicebridge" "1"
      $set "servicebridge.type" "REST"
       PROGRAM-ID. SONGS.
```

In this case, no specific configuration is required at compile time, so the compile command is just:

```
iscc SONGS.cbl
```

Web Service generation with isCOBOL IDE

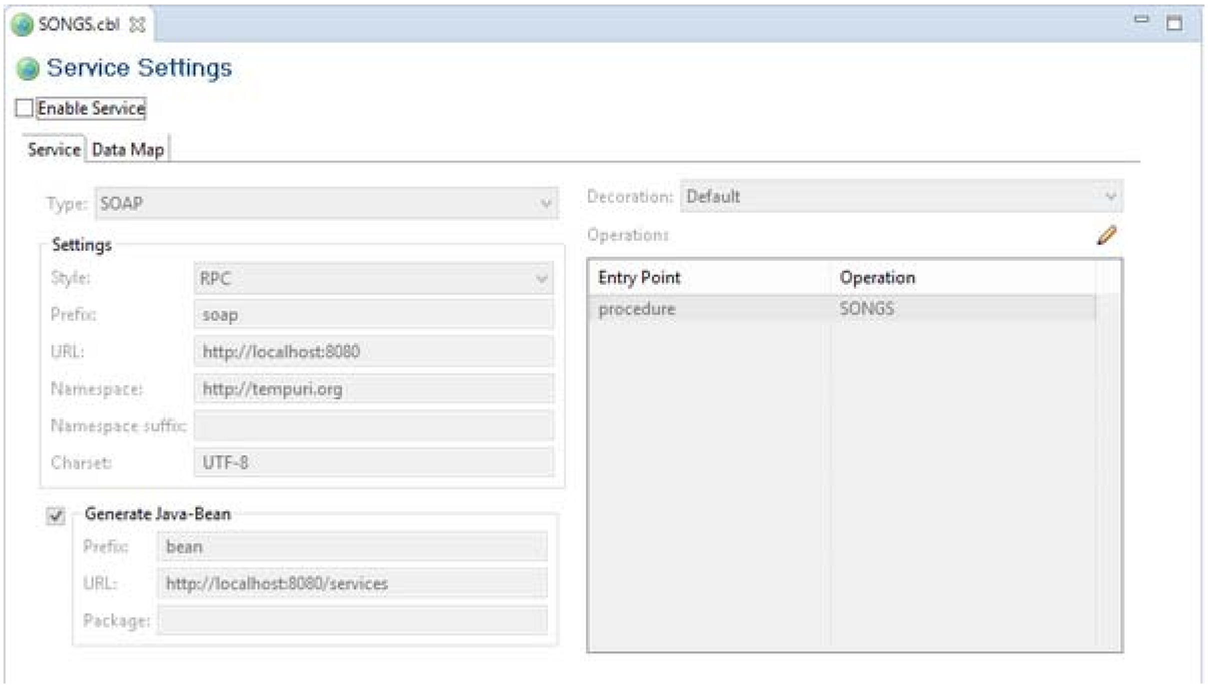
In order to be able to easily test our Web Service at the end of the process, create a isCOBOL Project.

1. Click *File* in the menu bar
2. Select *New*
3. Choose *isCOBOL Project*

Once the new project is created

1. Copy the SONGS.cbl source file to the *source* folder of the project
2. Right click on SONGS.cbl
3. Select *Open With*
4. Choose *isCOBOL Service Editor*

The following editor will open



Type: SOAP

Settings

Style: RPC

Prefix: soap

URL: http://localhost:8080

Namespace: http://tempuri.org

Namespace suffix:

Charset: UTF-8

Generate Java-Bean

Prefix: bean

URL: http://localhost:8080/services

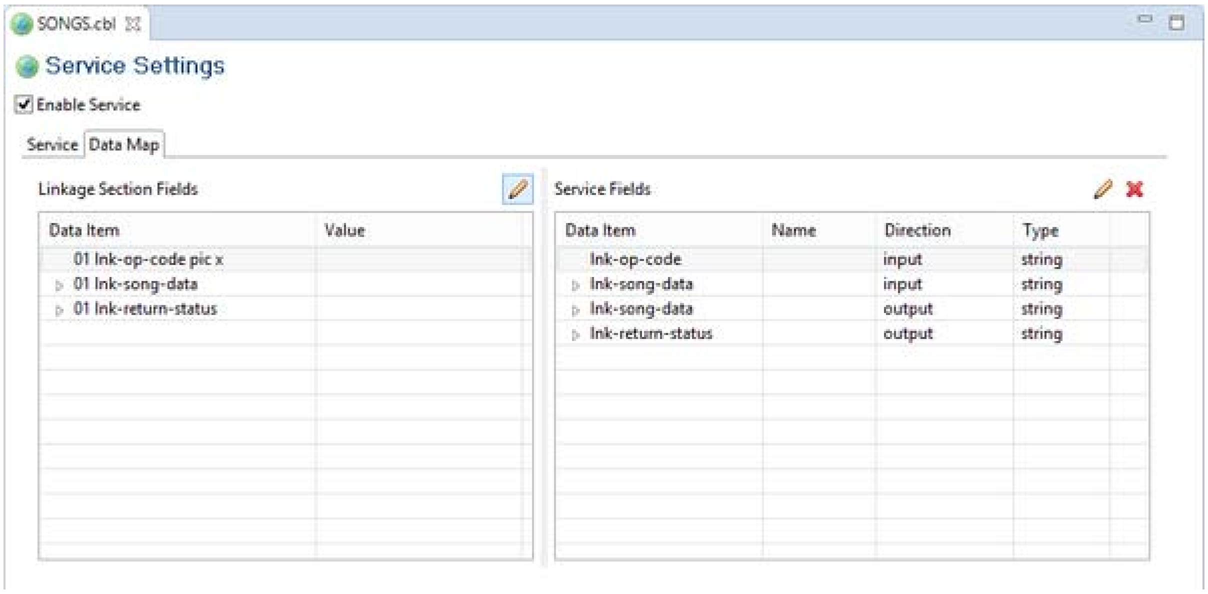
Package:

Decoration: Default

Operations

Entry Point	Operation
procedure	SONGS

1. Check the option *Enable Service*
   If no action is taken, the Compiler generates a Web Service parameter for each elementary COBOL data item and the parameter is assumed to be input/output. In this case instead we wish to define the first group item as input, the second as i/o, and the third as output, because the archive record buffer is shared between caller and callee, while the other two parameters are one-way, so switch to the Data Map tab and
2. Delete *lnk-return-status | input* from the *Service Fields* list as we want this field only as output
3. Delete *lnk-op-code | output* from the *Service Fields* list as we want this field only as input



Linkage Section Fields

Data Item	Value
01 lnk-op-code pic x	
01 lnk-song-data	
01 lnk-return-status	

Service Fields

Data Item	Name	Direction	Type
lnk-op-code		input	string
lnk-song-data		input	string
lnk-song-data		output	string
lnk-return-status		output	string

As soon as you save modification in this editor, the SONGS.cbl source file is automatically updated as follows:

- the directive *$set "servicebridge" "1"* is added at the top of the source file
- the directive *$elk input* is added on top of the lnk-op-code group item
- the directive *$elk output* is added on top of the lnk-returns-status group item

At this point you can compile SONGS.cbl.

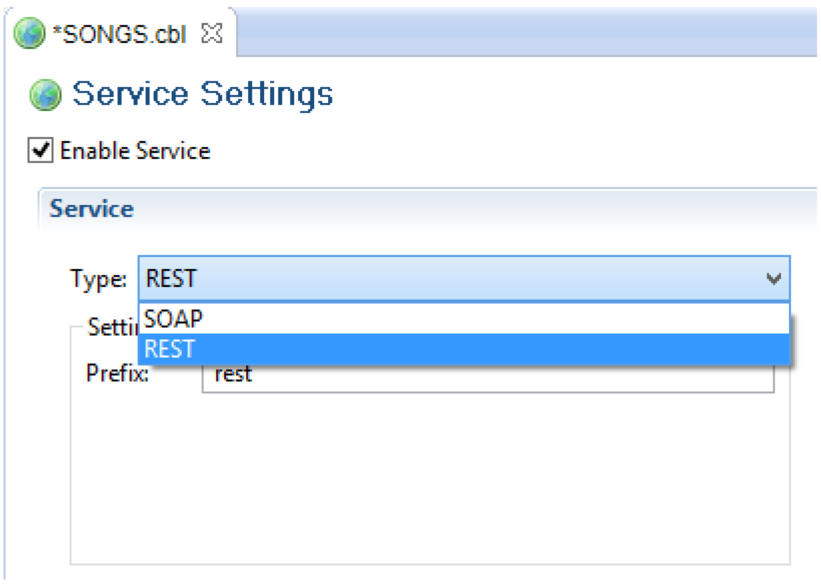
At the end of the compilation process, you will find two additional files in your project:

- *source/soapSONGS.cbl* : the bridge program that allows our program to be called as Web Service, and
- *output/SONGS.wsdl* : the XML descriptor of the service

You can start your Web Service with the following steps:

- Right click on the project name
- Select *Run As*
- Choose *isCOBOL EIS Servlet*

We've just demonstrated how to create a SOAP Web Service. With the next step, we're going to generate a REST Web Service instead. To achieve it, return to the Service Editor and change the field *Type* from SOAP to REST.



Type: REST

Prefix: rest

Note - By switching the Type value, the Settings frame content changes, allowing to set only the settings that are suitable for the selected service type.

After it, save modification and recompile SONGS.cbl to obtain the following new item in the project *source* folder:

- *restSONGS.cbl* : the bridge program that allows our songs program to be called as REST Web Service.

You can start your Web Service with the following steps:

- Right click on the project name
- Select *Run As*
- Choose *isCOBOL EIS Servlet*

Service Editor fields

In this section we map all the fields of the Service Editor with the corresponding Compiler property or directive that will be generated when you save modification.

Service Editor field	**Corrsponding Compiler property/directive**
Decoration	DECORATION Directive
Direction	INPUT Directive OUTPUT Directive
Enable Service	iscobol.compiler.servicebridge (boolean)
Generate Java-Bean / Prefix	iscobol.compiler.servicebridge.bean.prefix
Generate Java-Bean / URL	iscobol.compiler.servicebridge.bean.url
Name	NAME Directive
Namespace	iscobol.compiler.servicebridge.soap.namespace
Operations	OPERATION Directive
Package	iscobol.compiler.servicebridge.bean.package
Settings / Prefix	iscobol.compiler.servicebridge.rest.prefix iscobol.compiler.servicebridge.soap.prefix
Settings / URL	iscobol.compiler.servicebridge.soap.url
Style	iscobol.compiler.servicebridge.soap.style
Type	iscobol.compiler.servicebridge.type
Use Group	USE GROUP Directive
Value	VALUE Directive

Customizing bridge programs through tagged areas

The SOAP and REST bridge program source files include tagged areas.

A tagged area is a block of code included between a start comment and an end comment, e.g.

```
*>start {iscobol}accept-http

   set request-method to comm-area:>getMethod()
   if request-method = "POST"
      comm-area:>acceptEx(request-varin)
   else
     comm-area:>accept(PAR-invar)
   end-if

*>end {iscobol}accept-http
```

Once the bridge program source file has been created, only the code inside tagged areas is regenerated by the Compiler.

If you need to add custom code to a bridge program, edit the bridge program source file and write your code outside of tagged areas, so before a start comment or after an end comment, e.g.

```
*>start {iscobol}accept-http

   set request-method to comm-area:>getMethod()
   if request-method = "POST"
      comm-area:>acceptEx(request-varin)
   else
     comm-area:>accept(PAR-invar)
   end-if

*>end {iscobol}accept-http
  < <  your code here > >
*>start {iscobol}http-to-linkage
    move  PAR-in to intPAR;;
```

SOAP runtime configuration

The following configuration is available for SOAP web services at run time.

WSDL Location

The download of the WSDL file can be achieved using a URL such as:

```
http://localhost:8080/test/servlet/SONGS?wsdl
```

where test is the webapp name and SONGS is the web service name. The download is requested via the ?wsdl parameter.

On the server the property iscobol.soap.wsdl.location controls the location of the wsdl file. It should point to a file system path where wsdl files are copies, ie:

```
iscobol.soap.wsdl.location=/opt/tomcat8/wsdl_files
```

The servlet appends the webservice name and ".wdsl" to this path to form a path name, which, if found, is then downloaded. If the file is not found or the property is not set, and http error 404 is returned.

Logging

Logging can be enabled globally and exceptions can be set on a per-method basis.

The following properties can be added to *iscobol.properties* to control SOAP services logging:

- iscobol.soap.log=0/1 enables or disables logging on a global level
- iscobol.soap.log.{methodname}=0/1 enables or disables logging for the SOAP service with the specified method name. This overrides the global settings specified above.
- iscobol.soap.log.folder= is used to configure the folder where log files should be generated. The file name is generated dynamically using the following pattern: *{methodname}-{SESSIONID}.log*.

For example: to enable logging for the SONGS web service add the following property:

```
iscobol.soap.log.songs=1
```

To enable logging for all SOAP web services except for the SONGS web service use the following:

```
iscobol.soap.log=1
iscobol.soap.log.songs=0
```

The log includes a trace of the HTTP request and response. Both header and body are traced. If a runtime error occurs during the service activity, such error is traced in the log as well.

If a SOAP envelope cannot be successfully extracted from the request, an exception log is written on standard error (catalina.out if using Tomcat). In the log there will be the exception message and the HTTP request data.

REST runtime configuration

The following configuration is available for REST web services at run time.

Logging

Logging can be enabled globally and exceptions can be set on a per-method basis.

The following properties can be added to *iscobol.properties* to control REST services logging:

- iscobol.rest.log=0/1 enables or disables logging on a global level
- iscobol.rest.log.{methodname}=0/1 enables or disables logging for the REST service with the specified method name. This overrides the global settings specified above.
- iscobol.rest.log.folder= is used to configure the folder where log files should be generated. The file name is generated dynamically using the following pattern: *{methodname}-{SESSIONID}.log*.

For example: to enable logging for the SONGS web service add the following property:

```
iscobol.rest.log.songs=1
```

To enable logging for all REST web services except for the SONGS web service use the following:

```
iscobol.rest.log=1
iscobol.rest.log.songs=0
```

The log includes a trace of the HTTP request and response. Both header and body are traced. If a runtime error occurs during the service activity, such error is traced in the log as well.

The request body is always logged as a string, so if the content is not in text format (e.g. content-type is "application/x-www-form-urlencoded"), then unreadable characters may appear in the log file.

Web Service Beans

The Service Bridge facility can also generate bean classes that can be used client side for test purposes.

Command-line usage

In order to enable this feature when compiling from the command-line, the configuration property iscobol.compiler.servicebridge.bean must be set to the same value as iscobol.compiler.servicebridge.type.

The following configuration entries demonstrate how to generate bean classes for a REST Web Service.:

```
iscobol.compiler.servicebridge=1
iscobol.compiler.servicebridge.type=REST
iscobol.compiler.servicebridge.bean=REST
```

With this kind of configuration, at the end of the compilation process, you will find two folders: "bean" and "servicebridge".

The "bean" folder contains:

- *beanSONGS.cbl* : the bean that allows to test the REST Web Service
- *beanSONGS.cpy*: copybook used by beanSONGS.cbl
- *beanSONGS.wrk*: copybook used by beanSONGS.cbl

The "servicebridge" folder contains:

- *restSONGS.cbl* : the bridge program that allows our songs program to be called as REST Web Service.

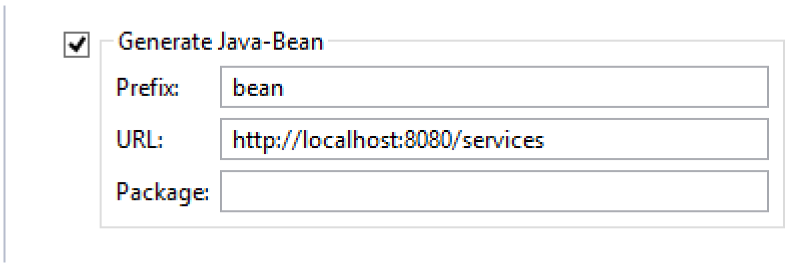
Refer to the installed sample README file for instructions about the deployment and testing of these items.

The same result can be obtained by setting the necessary properties directly in the source code, using the SET Directive. For example:

```
      $set "servicebridge" "1"
      $set "servicebridge.type" "REST"
      $set "servicebridge.bean" "REST"
       PROGRAM-ID. SONGS.
```

Usage in the IDE

In order to enable this feature within the isCOBOL Service Editor, just check the option *Generate Java-Bean*.


[✓] Generate Java-Bean

Prefix: bean

URL: http://localhost:8080/services

Package:

Bean structure

The bean class exposes the following methods:

get<parameter_name>()	Returns the value of the given parameter. *parameter_name* doesn't necessarily match the COBOL data item. This name is affected by ELK Directives.

set<parameter_name>()	Sets the value of the given parameter. *parameter_name* doesn't necessarily match the COBOL data item. This name is affected by ELK Directives.

The above pair of methods is repeated for each Linkage Section item. Other methods exposed:

get_url()	Returns the URL to which the bean is going to connect
run (<parameters>)	Executes the call to the Web Service passing the received parameters. The number of parameters required matches with the number of Linkage Section items expected by the COBOL program server side.
run()	Executes the call to the Web Service passing the parameters that were previously set by invoking *set<parameter_name>()*
set_url()	Sets the URL to which the bean is going to connect

The bean source code also includes two copybooks that allow you to include custom code.

<beanPrefix><programName>.wrk	Copybook that hosts custom data items.
<beanPrefix><programName>.cpy	Copybook that hosts custom procedural code.

By relying on these copybooks you can invoke *set<parameter_name>()* to set the parameters for the Web Service and then invoke *run()*. In this way you obtain a stand alone COBOL program that can consume the Web Service.

COBOL approach using REST

In order to exchange data through a web service, you need to set up a server-side requestor, or 'producer' program, as well as a client-side provider, or 'consumer' program, as explained in the Introduction. The REST-style architecture is stateless, with the client and server sides running separately. They communicate using HTTP protocol, and REST supports messages in JSON, XML, and other formats. The following chapters describe these two different types of COBOL programs.

COBOL REST producer

In order to develop a COBOL REST producer (server-side), to process requests and provide data, the COBOL program has to be transformed to be executed like a Web Service REST. This objective is achieved through the *HTTPHandler* class that allows communication with HTML pages through AJAX retrieving data and printing results.

In the isCOBOL sample folder you will find the folder eis/webservices/rest that contains an example of a COBOL REST producer (REST Web Service) and an example of a COBOL REST consumer to be used to test the service.

In the server folder there is a COBOL source program called "ISFUNCTIONS.cbl" that exposes two services: ISFUNCTION_GETZIP and ISFUNCTION_GETCITY that allow searching a US city name by zip code or by name.

This program has three entries:

- MAIN, the default entry where the values to be used are loaded from the JSON stream:

```
           move "94101"          to  a-zipcode(1).
           move "San Francisco" to  a-city(1).
           move "San Francisco" to  a-county(1).
           move "California"    to  a-state(1).

           move "92123"          to  a-zipcode(2).
           move "San Diego"     to  a-city(2).
           move "San Diego"     to  a-county(2).
           move "California"    to  a-state(2).

           move "10001"          to  a-zipcode(3).
           move "New York"      to  a-city(3).
           move "New York"      to  a-county(3).
           move "New York"      to  a-state(3).

           move "89044"          to  a-zipcode(4).
           move "Las Vegas"     to  a-city(4).
           move "Clark"         to  a-county(4).
           move "Nevada"        to  a-state(4).

           move "Program Loaded" to ok-message;;
           comm-area:>displayJSON (ok-page).
           goback.
```

- ISFUNCTION_GETZIP, a COBOL entry point that receives into *isfunction-getZipCode* working storage structure, a name of a US city and returns the zip code into *isfunction-returnZipCode* as JSON stream using *displayJSON()* method:

```
           entry "ISFUNCTION_GETZIP" using comm-area.

           comm-area:>accept (isfunction-getZipCode).

           move 1 to idx.
           search array-data varying idx
              at end
                 move "Zip code not Found"  to returnZipCode
              when city-zipCode = a-city(idx)
                 move a-zipcode(idx)  to returnZipCode
           end-search.

           comm-area:>displayJSON (isfunction-returnZipCode).

         goback.
```

where *isfunction-getZipCode* working storage structure is defined like:

```
       01  isfunction-getZipCode identified by "".
           03 identified by "get_Zip_Code".
              05 city-zipCode  pic x any length.
```

and *isfunction-returnZipCode* working storage structure is defined like:

```
       01  isfunction-returnZipCode identified by "".
           03 identified by "Zip_Code".
              05 returnZipCode  pic x any length.
```

- ISFUNCTION_GETCITY, a COBOL entry point that receives into *isfunction-getCity* working storage structure, a zip code of a US city and return the city name into *isfunction-retrievedCity* variable as JSON stream using displayJSON() method:

```
           entry "ISFUNCTION_GETCITY" using comm-area.

           comm-area:>accept (isfunction-getCity).

           move 1 to idx.
           search array-data varying idx
              at end
                 move "City not Found"  to returnCity
              when zipCode-city(1:5) = a-zipcode(idx)
                 move a-city(idx)  to returnCity
           end-search.

           comm-area:>displayJSON (isfunction-retrievedCity).

           goback.
```

where *isfunction-getCity* working storage structure is defined like:

```
       01  isfunction-getcity identified by "".
             03 identified by "get_City".
                05 zipCode-city  pic x any length.
```

and *isfunction-retrievedCity* working storage structure is defined like:

```
       01  isfunction-retrievedCity identified by "".
           03 identified by "City".
              05 returnCity  pic x any length.
```

In order to have this ISFUNCTIONS.cbl working correctly, it should be deployed inside a Java Servlet container like Tomcat, WildFly (ex JBoss AS), IBM WebSphere or BEA WebLogic.

COBOL REST consumer

In order to develop a COBOL REST consumer (client-side), to invoke REST Web Service, the COBOL program should take advantage of *HTTPClient* class that allows it to communicate with COBOL REST producer entry points through HTTP protocol. Also to allow the definition of HTTP parameters, an *HTTPData.Params* class is provided.

In the isCOBOL sample folder you will find the folder eis/webservices/rest/client that contains an example of COBOL client program called "CLIENTH.cbl" to communicate with the "ISFUNCTIONS.cbl" server service described on the previous page.

This program invokes ISFUNCTION_GETZIP service to return the zip code of San Diego, and the ISFUNCTION_GETCITY service to return the name of the city whose zip code is 89044.

This program shows how to do the following necessary steps:

- Include HTTPClient and HTTPData.Params classes in COBOL repository:

```
       configuration section.
       repository.
           class http-client as "com.iscobol.rts.HTTPClient"
           class http-param  as "com.iscobol.rts.HTTPData.Params".
```

- Establish the connection with REST Web Service using doGet() method and checking the success of the operation using getResponseCode() method:

```
           http:>doGet("http://127.0.0.1:8080/isfunctions/servlet/
isCobol(ISFUNCTIONS)")
           http:>getResponseCode (response-code)
```

- Prepare the city name as parameter to be pass to the service

```
          move "San Diego" to city-zipCode.
          set params = http-param:>new():>add("get_Zip_Code", city-zipCode).
```

- Invoke the ISFUNCTION_GETZIP with prepared parameter and getting back the zip code:

```
           http:>doGet ("http://127.0.0.1:8080/isfunctions/"
                        "servlet/isCobol(ISFUNCTION_GETZIP)", params)

           http:>getResponseCode (response-code)
           if response-code = 200
              http:>getResponseJSON (isfunction-retrievedZipCode)
```

where isfunction-retrievedZipCode working storage structure is defined like:

```
       01  isfunction-retrievedZipCode identified by "".
            03 identified by "Zip_Code".
               05 zipCode  pic x any length.
```

and 92123 is the zip code of San Diego saved into zipCode COBOL variable.

A similar approach is followed to provide the city name and receive it's zip code.

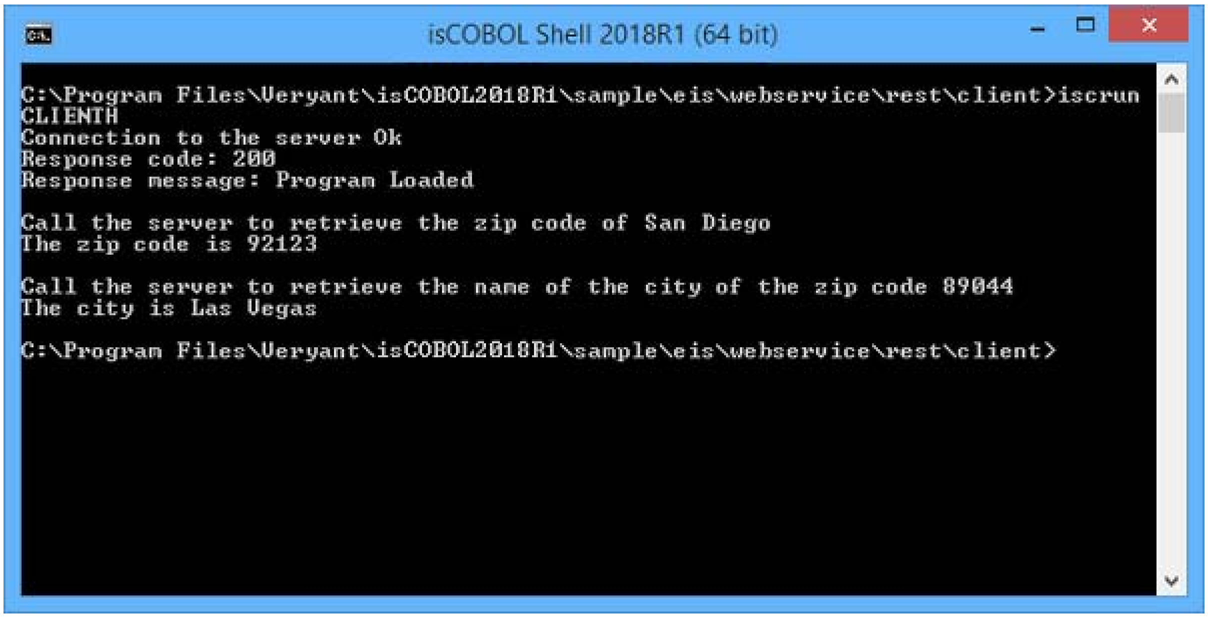
- Compile the program with the command:

```
iscc client.cbl
```

- and run it with the command:

```
iscrun CLIENTH
```

this is the result:



```
C:\Program Files\Veryant\isCOBOL2018R1\sample\eis\webservice\rest\client>iscrun
CLIENTH
Connection to the server Ok
Response code: 200
Response message: Program Loaded

Call the server to retrieve the zip code of San Diego
The zip code is 92123

Call the server to retrieve the name of the city of the zip code 89044
The city is Las Vegas

C:\Program Files\Veryant\isCOBOL2018R1\sample\eis\webservice\rest\client>
```

COBOL approach using SOAP

EIS is provided as preliminary support to develop COBOL programs capable to consume SOAP Web Services based on XML.

COBOL SOAP consumer

In order to develop a COBOL SOAP consumer (client-side), to invoke a SOAP Web Service, the COBOL program should take advantage of *HTTPClient* class. This class contains several useful methods to work with SOAP Web Service.

In the isCOBOL sample folder you find the folder eis/webservices/soap/client that contains an example of COBOL client program called "IP2GEO.cbl" that shows how to use a SOAP Web Service available over the internet at http://ws.cdyne.com/ip2geo/ip2geo.asmx .

This service "Resolves IP addresses to Organization, Country, City, and State/Province, Latitude, Longitude. In most US cities it will also provide some extra information such as Area Code, and more."

A SOAP Web Service usually provides a way to inquire about the functionality available and the parameters that should be used.

The isCOBOL STREAM2WRK command line utility simplifies the creation of the working storage definitions needed to manage the XML envelope. STREAM2WRK reads the WSDL definition from a URL, with the ?WSDL appended to it. For example, http://ws.cdyne.com/ip2geo/ip2geo.asmx?WSDL was used to generate this working storage:

```
      *> binding name=ResolveIP, style=document
       01  soap-in-ResolveIP identified by 'soapenv:Envelope'.
           03 identified by 'xmlns:soapenv' is attribute.
            05 filler pic x(39) value 'http://www.w3.org/2003/05/soap-envelope'.
           03 identified by 'xmlns:tns' is attribute.
            05 filler pic x(20) value 'http://ws.cdyne.com/'.
           03 identified by 'soapenv:Body'.
               06 identified by 'tns:ResolveIP'.
                07 identified by 'tns:ipAddress'.
                 08 a-ipAddress pic x any length.
                07 identified by 'tns:licenseKey'.
                 08 a-licenseKey pic x any length.

       01  soap-out-ResolveIP identified by 'Envelope'
                 namespace 'http://www.w3.org/2003/05/soap-envelope'.
           03 identified by 'Body'.
               06 identified by 'ResolveIPResponse'
                  namespace 'http://ws.cdyne.com/'.
                07 identified by 'ResolveIPResult'.
                 08 identified by 'City'.
                  09 a-City pic x any length.
                 08 identified by 'StateProvince'.
                  09 a-StateProvince pic x any length.
                 08 identified by 'Country'.
                  09 a-Country pic x any length.
                 08 identified by 'Organization'.
                  09 a-Organization pic x any length.
                 08 identified by 'Latitude'.
                  09 a-Latitude pic s9(16)v9(2).
                 08 identified by 'Longitude'.
                  09 a-Longitude pic s9(16)v9(2).
                 08 identified by 'AreaCode'.
                  09 a-AreaCode pic x any length.
                 08 identified by 'TimeZone'.
                  09 a-TimeZone pic x any length.
                 08 identified by 'HasDaylightSavings'.
                  09 a-HasDaylightSavings pic x any length.
                 08 identified by 'Certainty'.
                  09 a-Certainty pic s9(5).
                 08 identified by 'RegionName'.
                  09 a-RegionName pic x any length.
                 08 identified by 'CountryCode'.
                  09 a-CountryCode pic x any length.
```

The IP2GEO program will invoke the *ResolveIP* functionality providing the IP address and receiving some geographic information like City, State, Country, etc.

This program shows how to do the following necessary steps::

- Include HTTPClient class in COBOL repository:

```
       configuration section.
       repository.
           class http-client as "com.iscobol.rts.HTTPClient"
```

- Include the working storage definition to use XML envelope generated from WSDL by STREAM2WRK utility:

```
       WORKING-STORAGE SECTION.
       copy "ip2geo.cpy".
```

- Provide the IP address to obtain information and call the ResolveIP service using doPostEx() method passing the URL of service, the SOAP media type and the input envelope generated from STREAM2WRK for ResolveIP service:

```
           move "209.235.175.10"   to a-ipAddress
           http:>doPostEx (
                           "http://ws.cdyne.com/ip2geo/ip2geo.asmx"
                           "text/xml; charset=utf-8"
                           soap-in-ResolveIP).
```

  Note - The type "text/xml;charset=utf-8" is suitable for SOAP v1.1. If the service is SOAP v1.2, use "application/soap+xml; charset=utf-8" instead.

- Check the response if successful and show results:

```
           http:>getResponseCode (response-code).
           display "Response code: " response-code.
           if response-code = 200
              http:>getResponseXML (soap-ResolveIP-output)
              display "City=" a-city
              display "StateProvince=" a-stateProvince
              display "Country=" a-country
              display "Organization=" a-Organization
              display "Latitude=" a-latitude
              display "Longitude=" a-longitude
              display "AreaCode=" a-areaCode
              display "TimeZone=" a-timeZone
              display "Daylight savings=" a-HasDaylightSavings
              display "Certainty=" a-certainty
              display "RegionName=" a-regionName
              display "CountryCode=" a-countryCode
```

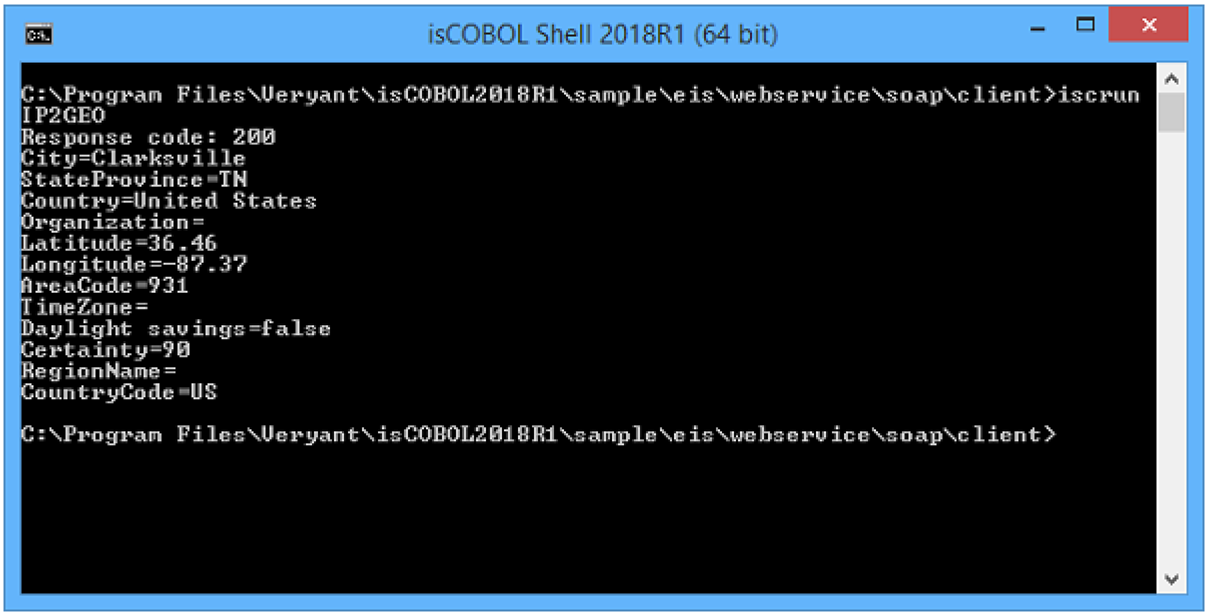
Compile the program with the command:

```
iscc IP2GEO.cbl
```

and run it with the command:

```
iscrun IP2GEO
```

This is the result of execution of IP2GEO that consumes the ResolveIP SOAP Web Service:



```
C:\Program Files\Veryant\isCOBOL2018R1\sample\eis\webservice\soap\client>iscrun
IP2GEO
Response code: 200
City=Clarksville
StateProvince=TN
Country=United States
Organization=
Latitude=36.46
Longitude=-87.37
AreaCode=931
TimeZone=
Daylight savings=false
Certainty=90
RegionName=
CountryCode=US

C:\Program Files\Veryant\isCOBOL2018R1\sample\eis\webservice\soap\client>
```

Chapter 3

Authentication and Authorization method

Introduction

You can obtain limited access to an HTTP Service taking advantage of existing Authentication and Authorizations providers like Google and Facebook based on OAuth 2.0 standard.

OAuth 2.0 is an open protocol to allow secure authorization in a simple and standard method from web, mobile and desktop applications. The request is to make a COBOL program accessible only by the logged users without checking for each single program.

Servlet Container Configuration

Servlet containers (e.g. Apache Tomcat) have fully configurable authentication systems, however they usually don't fit well with the authentication from another server, thus they are not used in this example.

You need to define a safe area where the isCOBOL application can be invoked only after a successful authentication. Since the isCOBOL applications are executed as if they belonged to the same session, you can set an environment property after the authentication process and then check for it each time an application runs. However it is not handy or safe to put a check in each program, thus you can define a filter that does this job.

The configuration file *web.xml* will therefore contain the following entries:

```
<filter>
  <filter-name>isCOBOL security</filter-name>
  <filter-class>SecurityFilter</filter-class>
</filter>
<filter-mapping>
   <filter-name>isCOBOL security</filter-name>
   <url-pattern>/servlet/*</url-pattern>
</filter-mapping>
```

In this way you specify a program to run before running any program located under the URL /servlet. This program could be the following isCOBOL class:

```
       class-id. SecurityFilter as "SecurityFilter"
                                       implements c-filter.
       configuration section.
       repository.
           class j-ioexception as "java.io.IOException"
           class c-filter as "javax.servlet.Filter"
           class c-filter-chain as "javax.servlet.FilterChain"
           class c-filter-config as "javax.servlet.FilterConfig"
           class c-ServletException as "javax.servlet.ServletException"
           class c-ServletRequest as "javax.servlet.ServletRequest"
           class c-ServletResponse as "javax.servlet.ServletResponse"
           class c-HttpServletResponse as
                             "javax.servlet.http.HttpServletResponse"
           class c-HttpServletRequest as
                             "javax.servlet.http.HttpServletRequest"
           .
       id division.
       object.
       data division.
       working-storage section.

       procedure division.

       id division.
       method-id. init as "init".
       linkage section.
       77  cfg object reference c-filter-config.
       procedure division using cfg raising c-ServletException.
       main.
       end method.

       id division.
       method-id. c-destroy as "destroy".
       procedure division.
       main.
       end method.
       id division.
       method-id. doFilter as "doFilter".
       working-storage section.
       77  email pic x any length.
       77  uri pic x any length.
       77  http-response  object reference c-HttpServletResponse.
       linkage section.
       77  request object reference c-ServletRequest.
       77  response object reference c-ServletResponse.
       77  f-chain object reference c-filter-chain.
       procedure division using request response f-chain
                        raising c-ServletException j-IOException.
       main.
           accept email from environment "openid.email".
           if email = space
              set http-response to response as c-HttpServletResponse
              http-response:>sendError
                         (c-HttpServletResponse:>SC_FORBIDDEN)
           else
              f-chain:>doFilter (request response)
           end-if.
       end method.
       end object.
```

This program simply checks if the property "openid.email" has been set to a value different from space and in that case it forwards the execution to the next filter in the chain, otherwise it stops the execution with an error code.

This assures you that any program under the URL /servlet, the safe area, will be executed only if previously in the same session, some program has set the property.

You now need to write that program and define it outside the safe area.

Facebook Authentication

Here we show an example of how to implement a program in order to authenticate the access using the Facebook authentication. You can find Facebook's documentation at the address: https://developers.facebook.com/docs/facebook-login/manually-build-a-login-flow/v2.0.

This kind of authentication requires your program to redirect the login phase to the Facebook site and then performs some HTTP requests to the Facebook APIs. Your program will use the following classes:

```
           configuration section.
           repository.
              class web-area as "com.iscobol.rts.HTTPHandler"
              class http-client as "com.iscobol.rts.HTTPClient"
              class http-params as "com.iscobol.rts.HTTPData.Params"
              class j-bigint as "java.math.BigInteger"
              class j-securernd as "java.security.SecureRandom"
              .
           working-storage section.
           01  params object reference http-params.
           01  http object reference http-client.
```

The classes j-bigint and j-securernd are used to create a secure random number whose purpose will be explained later.

In order to use the Facebook authentication, you need a Facebook App ID that you can create and retrieve on the App Dashboard (https://developers.facebook.com/apps/).

There you get a client ID and a client secret that are necessary in the authentication process.

Let's say that the URL of our program is “http://veryant.com/oauth/FBConnect”, then the WORKING-STORAGE SECTION will contain:

```
       78  client-id value "<client-id-by-Facebook>".
       78  clsc value "<client-secret-by-Facebook>".
       78  redir value "http://veryant.com/oauth/FBConnect".
       78  realm value "http://veryant.com/oauth".
       01  state pic x any length.
```

The login process can be divided in three stages:

- Request the authentication from Facebook through a redirection;
- Get the authentication data in order to be able to query Facebook APIs;
- Get the logged user data.

The program is called two times: the first time by the user in order to start the authentication process and the second time by a Facebook redirection.

The first phase is simply a redirection where you specify what URL must be called back.

You must protect the security of your users by preventing request forgery attacks. In order to be sure that this callback is performed by the URL you actually called, a random id (state token) must be supplied. According to Google documentation (https://developers.google.com/accounts/docs/OAuth2Login): "One good choice for a state token is a string of 30 or so characters constructed using a high-quality random-number generator". These tokens are often referred to as cross-site request forgery (CSRF) tokens.

You can create this secure random id using the classes j-securernd and j-bigint as in following code:

```
           set state=j-bigint:>new(130 j-securernd:>new):>toString(32).
```

The code for redirection then will be:

```
       phase-1-redirection.
             set state=j-bigint:>new(130 j-securernd:>new):>toString(32).
             set params = http-params:>new
                      :>add ("client_id" client-id)
                      :>add ("display" "popup")
                      :>add ("response_type" "code")
                      :>add ("scope" "email")
                      :>add ("redirect_uri" redir)
                      :>add ("state" state)

             comm-area:>redirect (
               "https://www.facebook.com/dialog/oauth" params).
```

The second phase begins when the same application is called back by Facebook, as specified by the redir variable. The program can easily tell if it is the first run or the second by the setting of the variables state and http-state: the former is set by phase 1 while the latter will be passed by Facebook in the redirection of the login. So the initial part of the program could be the following one:

```
       linkage section.
       01  comm-area object reference web-area.
       procedure division using comm-area.
       main.

         accept client-id from environment "app_id_by_fb"
         accept clsc   from environment "app_secret_by_fb".

         accept redir from environment "realdir_fb".

         if user-email = ""
           perform do-auth
         else
           perform run-first-program
         end-if.
         goback.

       do-auth.
         initialize http-response.
         comm-area:>accept(http-response).
         if http-state = space
           perform phase-1-redirection
         else
           if http-state = state
              perform phase-2-get-auth-token
              perform phase-3-get-info
              perform set-first-program
              perform run-first-program
           else
              string  "Forged state! (" http-state ")(" state ")"
                      into err-msg
              comm-area:>displayError(403 err-msg)
           end-if
         end-if.
```

The parameters received by Facebook are described in the following variable:

```
    01  http-response identified by "_".
       03 identified by "state".
         05 http-state  pic x any length.
       03 identified by "code".
         05 http-code  pic x any length.
```

The parameter code (stored in http-code) is the one you need in order to get the authorization to query the Facebook APIs, along with your client ID and client secret. The source code of the second phase could be the following:

```
       phase-2-get-auth-token.
            set http = http-client:>new
            set params = http-params:>new
                :>add ("code" http-code)
                :>add ("client_id" client-id)
                :>add ("client_secret" clsc)
                :>add ("redirect_uri" redir)
                :>add ("grant_type" "authorization_code")
            try
              http:>doPost (
               "https://graph.facebook.com/oauth/access_token" params)
              http:>getResponseCode (response-code)
              if response-code = 200
                 http:>getResponseJSON (fb-token)
              else
                 comm-area:>displayError(response-code "")
                 goback
              end-if
            catch exception
              comm-area:>displayError(500 exception-object:>toString)
              goback
            end-try.
```

The fb-token data item is defined as follows:

```
01  fb-token  identified by "".
    03 access_token identified by "access_token".
       05 access_token-data pic x any length.
    03 token_type identified by "token_type".
       05 token_type-data pic x any length.
    03 expires_in identified by "expires_in".
       05 expires_in-data pic x any length.
```

If the request is successful, the program will receive in access-token-data a character string, called "access token", that allows you to call anything among the Facebook APIs. You still don't have any information about the person who is logged, so you need to get some basic information.

In the third phase you may choose to call the API "me": this API returns a JSON payload whose data is described in the following variable:

```
01  user-info identified by "".
    03 identified by "name".
       05 user-name  pic x any length.
    03 identified by "email".
       05 user-email  pic x any length.
    03 identified by "id".
       05 user-id  pic x any length.
```

The source code could be the following:

```
       phase-3-get-info.
           string "https://graph.facebook.com/me?"
                  "fields=name,email&"
                  "access_token=" access_token-data
                  into authorization
            set http = http-client:>new
            try
              http:>doGet (authorization)
              http:>getResponseCode (response-code)
              if response-code = 200
                 http:>getResponseJSON (user-info)
              else
                 comm-area:>displayError(response-code "")
                 goback
              end-if
            catch exception
              comm-area:>displayError(500 exception-object:>toString)
              goback
            end-try.
```

Note that this time there is a STRING command instead of passing the parameters in the usual way. This is because the access token must be passed as it is.

If the call is successful, then the only thing left to do is start the next program, i.e. the first program in the application, for example:

```
       set-first-program.
             set environment "openid.email" to user-email.
             accept data-dir from environment "file.prefix"
             string data-dir "/" user-email into data-dir

             call "c$makedir" using data-dir
             set environment "file.prefix" to data-dir.

          run-first-program.
             comm-area:>redirect ("_index.html").
```

Google Authentication

Here we show an example about how to implement a program in order to authenticate the access using Google authentication. You can find Google's documentation at the address: https://developers.google.com/accounts/docs/OAuth2Login.

This kind of authentication requires your program to redirect the login phase on the Google site and then performs some HTTP requests to the Google APIs. Your program will use the following classes:

```
configuration section.
repository.
  class web-area    as "com.iscobol.rts.HTTPHandler"
  class http-client as "com.iscobol.rts.HTTPClient"
  class http-params as "com.iscobol.rts.HTTPData.Params"
  class j-bigint    as "java.math.BigInteger"
  class j-securernd as "java.security.SecureRandom"
  .
working-storage section.

01  params object reference http-params.
01  http object reference http-client
```

The classes *j-bigint* and *j-securernd* are used to create a secure random number whose purpose will be explained later.

According to Google's documentation "Before your application can use Google's OAuth 2.0 authentication system for user login, you must set up a project in the Google Developers Console (https://console.developers.google.com/) to obtain OAuth 2.0 credentials, set a redirect URI, and (optionally) customize the branding information that your users see on the user-consent screen. You can also use the Developers Console to create a service account, enable billing, set up filtering, and do other tasks. For more details, see the Google Developers Console Help (https://developers.google.com/console/help/console)"

There you get a client ID and a client secret that will be necessary in the authentication process.

Let's say that the URL of our program is http://veryant.com/ismobile3/OpenIDConnect then the WORKING-STORAGE SECTION will contain:

```
78  client-id value "<client-id-by-Google>".
78  clsc value "<client-secret-by-Google>".
78  redir value "http://veryant.com/oauth/GOOGLEConnect".
78  realm value "http://veryant.com/oauth".
01  state pic x any length.
```

The login process can be divided in three stages:

- Request the authentication from Google through a redirection;
- Get the authentication data in order to be able to query Google APIs;
- Get the data about the logged user.

The program will be called two times: the first time by the user in order to start the authentication process, the second time by a Google redirection.

The first phase is simply a redirection in which you must specify what URL must be called back.

You must protect the security of your users by preventing request forgery attacks. In order to be sure that this callback is performed by the URL you actually called, a random id (state token) must be supplied. According to Google documentation: "One good choice for a state token is a string of 30 or so characters constructed using a high-quality random-number generator". These tokens are often referred to as cross-site request forgery (CSRF) tokens.

You can create this secure random id using the classes j-securernd and j-bigint as in following code:

```
        set state=j-bigint:>new(130 j-securernd:>new):>toString(32).
```

The code for redirection then will be:

```
       phase-1-redirection.
            set state to
                    j-bigint:>new(130 j-securernd:>new):>toString(32).
            set params = http-params:>new
                      :>add ("client_id" client-id)
                      :>add ("response_type" "code")
                      :>add ("scope" "openid email")
                      :>add ("redirect_uri" redir)
                      :>add ("state" state)
                      :>add ("openid.realm" realm)
            comm-area:>redirect ("https://accounts.google.com/o/oauth2/auth" params).
```

Note that the SCOPE parameter has the value "openid email": if you do not include "email" then the logger will not share his email address with your application.

The second phase begins when the same application is called back by Google, as specified by the redir variable. The program can easily tell if it is the first run or the second by the setting of the variables state and http-state: the former is set by phase 1 while the latter will be passed back by Google in the redirection of the login. So the initial part of the program could be the following:

```
       linkage section.
       01  comm-area object reference web-area.
       procedure division using comm-area.
       main.

         accept client-id from environment "client_id_by_google"
         accept clsc  from environment "client_secret_by_google".

         accept redir from environment "realdir".
         accept realm from environment "realm".


         if user-email = space
           perform do-auth
         else
           perform run-first-program
         end-if.
         goback.

       do-auth.
         initialize http-response.
         comm-area:>accept(http-response).
         if http-state = space
           perform phase-1-redirection
         else
           if http-state = state
              perform phase-2-get-auth-token
              perform phase-3-get-info
              perform set-first-program
              perform run-first-program
           else
              comm-area:>displayError(403 "Forged state!")
           end-if
         end-if.
```

The parameters received back by Google are described in the following variable:

```
    01  http-response identified by "_".
       03 identified by "state".
         05 http-state  pic x any length.
       03 identified by "code".
         05 http-code  pic x any length.
```

The parameter code (stored in http-code) is the one you need in order to get the authorization to query the Google APIs, along with your client ID and client secret. The source code of the second phase could be the following:

```
       phase-2-get-auth-token.
            set http = http-client:>new
            set params = http-params:>new
                :>add ("code" http-code)
                :>add ("client_id" client-id)
                :>add ("client_secret" clsc)
                :>add ("redirect_uri" redir)
                :>add ("grant_type" "authorization_code")
            try
              http:>doPost (
                      "https://accounts.google.com/o/oauth2/token"
                      params)
              http:>getResponseCode (response-code)
              if response-code = 200
                 http:>getResponseJSON (google-auth)
              else
                 comm-area:>displayError(response-code "")
                 goback
              end-if
            catch exception
              comm-area:>displayError(500 exception-object:>toString)
              goback
            end-try.
```

If the request is successful, the program will receive a JSON payload, containing two strings of characters called "access_token" and "token_type" that allow you to call anything among the Google APIs. This is the isCOBOL description of the JSON:

```
       01  google-auth identified by "_".
           03 identified by "access_token".
              05 access-token  pic x any length.
           03 identified by "token_type".
              05 token-type  pic x any length.
           03 identified by "expires_in".
              05 expires-in  pic 9(9).
           03 identified by "id_token".
              05 id-token  pic x any length.
```

In the third phase you may choose to call the API "userinfo": this API returns a JSON payload whose data are described in the following variable:

```
          01  user-info identified by "_".
            03 identified by "id".
              05 user-id  pic x any length.
            03 identified by "email".
              05 user-email  pic x any length.
            03 identified by "verified_email".
              05 user-verified-email  pic x any length.
            03 identified by "name".
              05 user-name  pic x any length.
            03 identified by "given_name".
              05 user-given-name  pic x any length.
            03 identified by "family_name".
              05 user-family-name  pic x any length.
            03 identified by "link".
              05 user-link  pic x any length.
            03 identified by "picture".
              05 user-picture  pic x any length.
            03 identified by "gender".
              05 user-gender  pic x any length.
```

You still don't have any information about the person who logged in, so you need to get some basic information.

In the third phase you may choose to call the API "userinfo": this API returns a JSON payload whose data are described in the following variable:

```
       01  user-info identified by "_".
           03 identified by "id".
              05 user-id  pic x any length.
           03 identified by "email".
              05 user-email  pic x any length.
           03 identified by "verified_email".
              05 user-verified-email  pic x any length.
           03 identified by "name".
              05 user-name  pic x any length.
           03 identified by "given_name".
              05 user-given-name  pic x any length.
           03 identified by "family_name".
              05 user-family-name  pic x any length.
           03 identified by "link".
              05 user-link  pic x any length.
           03 identified by "picture".
              05 user-picture  pic x any length.
           03 identified by "gender".
              05 user-gender  pic x any length.
```

In order to query the Google APIs you need to put an authorization property in the header of each request: the property key will be "Authorization" while the property value will be the concatenation of the "token_type" plus the "access_token" separated by a space character. The source code could be the following:

```
       phase-3-get-info.
            string token-type " " access-token into authorization
            try
              http:>setHeaderProperty ("Authorization" authorization)
              http:>doGet (
                     "https://www.googleapis.com/oauth2/v2/userinfo")
              http:>getResponseCode (response-code)
              if response-code = 200
                 http:>getResponseJSON (user-info)
              else
                 comm-area:>displayError(response-code "")
                 goback
              end-if
            catch exception
              comm-area:>displayError(500 exception-object:>toString)
              goback
            end-try.
```

If the call is successful, then the only thing left to do is start the next program, i.e. the first program in the application, for example:

```
       set-first-program.
            set environment "openid.email" to user-email.
            accept data-dir from environment "file.prefix"
            string data-dir "/" user-email into data-dir

            call "c$makedir" using data-dir
            set environment "file.prefix" to data-dir.

         run-first-program.
            comm-area:>redirect ("_index.html").
```

For Complete examples of Facebook and Google authentications see the installed samples under sample\eis\other\oauth.

Twitter Authentication

If you need to implement a program in order to access some Twitter APIs using the application-only authentication, the following will explain how to do it. Also the example shows how to read some Tweets once connected. You can find Twitter's documentation at the address: https://developer.twitter.com/en/docs/basics/authentication/overview/application-only.

In order to use this kind of authentication you need to have a configured application on Twitter to get a "Consumer Key" (or "API Key") and a "Consumer secret" (or "API Secret").

These two strings are basically equivalent to a login name and a password to be used in an HTTP Basic Authentication.

Your COBOL program will define at least 2 classes: the class for doing an HTTP connection and the class for passing parameters in the HTTP requests, e.g.:

```
          CONFIGURATION SECTION.
          REPOSITORY.
             class http-client as "com.iscobol.rts.HTTPClient"
             class http-params as "com.iscobol.rts.HTTPData.Params"
             .

          WORKING-STORAGE SECTION.
          77  http object reference http-client.
          77  parms object reference http-params.
```

So the first HTTP request will be a typical POST request using the Basic authentication and supplying the parameter "grant_type" whose value will be "client_credentials".

```
           set parms = http-params:>new
                    :>add ("grant_type", "client_credentials")

           set http = http-client:>new.
           http:>setAuth ("<Consumer-key-by-Twitter>"
                          "<Consumer-secret-by-Twitter>").
           try
              http:>doPost (
               "https://api.twitter.com/oauth2/token" parms)
              http:>getResponseCode (response-code)
```

The response to this request will be a JSON-encoded payload: if the response code is different from 200 (OK), the JSON payload will be something like the following:

```
          {"errors":[
               {"label":"authenticity_token_error","code":99,"message":
                    "Unable to verify your        credentials"}]}
```

while if the response will be 200 the JSON payload will be something like this:

```
      {"token_type":"bearer","access_token":
"AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA2FAAAAAAAAAAAAAAAAAAAA3DAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAA AAAAAA"}
```

In order to get the data from this payload you can define the following structure in isCOBOL:

```
       01  twitter-auth identified by "".
           03 identified by "token_type".
              05 token-type  pic x any length.
           03 identified by "access_token".
              05 access-token  pic x any length.
```

So you can get the two strings with something like:

```
            if response-code = 200
               http:>getResponseJSON (twitter-auth)
```

According to the official documentation, you must verify that the token type is "bearer" and then you can use the access token to call the APIs you need, allowed by this authentication method.

For example, you can implement the "user_timeline" API: in order to do this, we need to use the access token as "bearer" instead of the login/password used previously. The new method *setAuth (ICobolVar a)* of HTTPClient do exactly this. You can also pass all the supported parameters. See https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline for the full documentation. E.g.:

```
if token-type = "bearer"
   http:>setAuth (access-token)
   set parms = http-params:>new
                           :>add ("count", "20")
                           :>add ("screen_name", "VeryantCOBOL");;
   http:>doGet ("https://api.twitter.com/1.1/statuses/user_timeline.json" parms)
```

In this case you perform the GET request according to the official documentation. This request will return two different JSON payloads depending on the success of the call, but, differently from what happened in the previous API, it seems that the response code is 200 in any case. This means that you cannot know which isCOBOL structure you must use in order to get the data from the payload.

The two formats returned by the above API are very different: when there is an error the format is very similar to the one already seen above when the authorization fails. If the operation return successfully, however, the payload will be an array of objects, whose length depends on the "count" parameter, each one including about 100 fields (see https://developer.twitter.com/en/docs/tweets/timelines/api-reference/get-statuses-user_timeline for a complete description).

In our example we are interested only in few fields, so we have defined a structure like the following:

```
       01  twitter identified by space.
           03 array identified by space occurs dynamic
                                      capacity cnt.
              05 identified by "text".
                 07 twittext pic x any length.
              05 identified by "user".
                 07 identified by "screen_name".
                    09 screen-name pic x any length.
```

The first 03 item is the data we need for our application.

This is the full program:

```
       PROGRAM-ID. tweet.

       CONFIGURATION SECTION.
       REPOSITORY.
           class http-client as "com.iscobol.rts.HTTPClient"
           class http-params as "com.iscobol.rts.HTTPData.Params"
           .

       WORKING-STORAGE SECTION.
       77  http object reference http-client.
       77  parms object reference http-params.
       77  i int.
       77  some-text pic x any length.
       77  response-code pic 999.

       77  api-key     pic x any length.
       77  api-secret  pic x any length.

       01  twitter-auth identified by "".
           03 identified by "token_type".
              05 token-type  pic x any length.
           03 identified by "access_token".
              05 access-token  pic x any length.

       01  twitter identified by space.
           03 array identified by space occurs dynamic capacity cnt.
              05 identified by "text".
                 07 twittext pic x any length.
              05 identified by "user".
                 07 identified by "screen_name".
                    09 screen-name pic x any length.

       PROCEDURE DIVISION.
       MAIN.

           accept api-key from environment "api_key"
           accept api-secret from environment "api_secret"

           set parms = http-params:>new
                    :>add ("grant_type", "client_credentials")
           set http = http-client:>new.
           http:>setAuth (api-key api-secret)
           try
              http:>doPost (
               "https://api.twitter.com/oauth2/token" parms)
              http:>getResponseCode (response-code)
              if response-code = 200
                 http:>getResponseJSON (twitter-auth)
                 if token-type = "bearer"
                    http:>setAuth (access-token)
                    set parms = http-params:>new
                       :>add ("count", "20")
                       :>add ("screen_name", "VeryantCOBOL");;
                    http:>doGet ("https://api.twitter.com/1.1"-
                                 "/statuses/user_timeline.json" parms)
                    if response-code = 200
                       display "Connection OK Response code="
                                       response-code;;
                       http:>getResponseJSON (twitter)
                       perform show-results
                    else
                       display "Response code=" response-code;;
                       http:>getResponsePlain (some-text)
                       display some-text
                       goback
                    end-if
                 else
                    display "wrong token-type=" token-type
                 end-if
              else
                 display "Connection problem. Response code="
                                              response-code;;
                 http:>getResponsePlain (some-text)
                 display some-text
                 goback
              end-if
           catch exception
              display exception-object:>toString
              goback
           end-try.
           goback.

       show-results.
           display "Total number of Tweets [" cnt "]"
           perform varying i from 1 by 1 until i > cnt
              display "Tweet " i
              display "@" screen_name(i) ": " twittext (i)
           end-perform.
```

where "api_key" and "api_secret" are the "Consumer Key" (or "API Key") and a "Consumer secret" (or "API Secret") are retrieved from the configuration file.

Chapter 4

HTML5/CSS3 JS and JSON

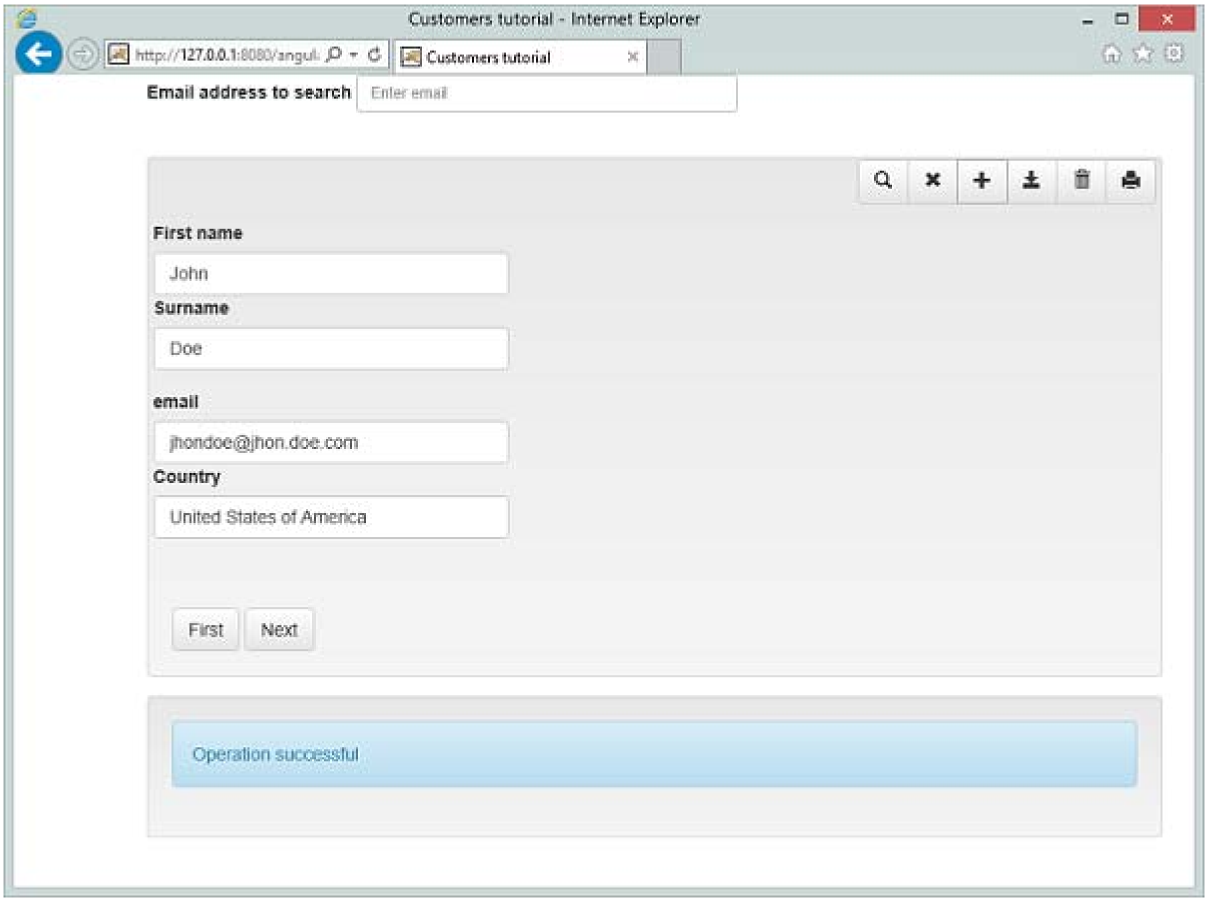
isCOBOL and AngularJS

With isCOBOL EIS taking advantage of COBOL REST producer and JSON COBOL integration, it is possible to write a Rich GUI Client Desktop application based on HTML5 and CSS3.

The client javascript library we recommend to work with is called AngularJS (http://angularjs.org/), and is developed and supported by Google.

This library, among other things, makes it easy to bind the data model coming from isCOBOL programs to the web page. An angular application is built on views and controllers. Each view (a page or part of a page) is handled by a controller, which fetches data from the isCOBOL servlet and binds it to the view's components.

The example page is described below and you’ll find all sources on sample/eis/other/angularjs.



Customers tutorial - Internet Explorer

http://127.0.0.1:8080/angul

Customers tutorial

Email address to search: Enter email

First name: John

Surname: Doe

email: jhondoe@jhon.doe.com

Country: United States of America

Operation successful

First of all we need to include all relevant javascript and CSS files in the head of HTML page:

```
<link href="css/bootstrap.min.css" rel="stylesheet" media="screen"/>
<link href="css/bootstrap-theme.min.css" rel="stylesheet" media="screen"/>
<link href="css/customers.css" rel="stylesheet" media="screen"/>
<script type="text/javascript" src="js/jquery.js"></script>
<script type="text/javascript" src="js/angular.min.js"></script>
<script type="text/javascript" src="js/bootstrap.min.js"></script>
<script type="text/javascript" src="js/app.js"></script>
```

Then we should indicate the tag ng-app in the <html> declaration in order to load the library and process all directives in the page:

```
<html ng-app="appTutorial">
```

All code to manage the HTML application is contained in the file *app.js*.

Let's examine it:

- It contains the application declaration (*angular.module('appTutorial', []);*)
- It defines a controller CustomersCtrl which will handle a customer page

The controller includes a *$scope* variable, which represents the current instance of the controller itself, and *$http*, which is an object provided by the AngularJS library that supports http requests to servers.

We define our model using *$scope.customer={}*, which creates an empty JSON object that will be filled by the isCobol program:

```
$scope.customer={};
```

Also, the controller defines methods to handle the buttons placed in the form, used for data navigation and processing. For example, the *getNextCustomer* method calls a COBOL entry point called AWEBX-NEXT that fetches the next customer in the dataset:

```
 $scope.getNextCustomer = function() {
       $http.get("servlet/isCobol(AWEBX_NEXT)")
           .then(function (response) {
               if (response.data._comm_buffer._status=="OK")
                   $scope.customer = response.data._comm_buffer;
           })
 }
```

Next, we need to bind the page, or a section of a page, to a controller. In this sample we bind the *CustomerCtrl* (Customer Controller) to a div, this means that each control inside the div will have access to the model and methods defined in the controller. The binding is done using the directive

```
<div class="container" ng-controller="CustomersCtrl">
```

inside the <div...> tag right after the body.

Inside the div we define an HTML form, which will be handled by the CustomerCtrl as well. This is done by specifying the directive ng-action="performSearch()" inside the form. Form submission will trigger the PerformSearch method of the controller:

```
<form ng-submit="performSearch()" class="form-inline" role="form">
```

Notice how each button in the form is bound to a method in the controller, meaning that the click will be handled by the method specified in the ng-click directive. Each INPUT tag in the form is bound to one of the data model defined in the controller. In this tutorial we only have one model: customer.

The structure of this model is defined in the AWEBX.cbl COBOL source code. Each time AWEBX.cbl is executed it returns the "response" record, which contains status about the performed operation and a customer record:

```
         01 comm-buffer identified by "_comm_buffer".
            03  filler identified by "_status".
               05 response-status pic x(2).
            03  filler identified by "_message".
               05 response-message pic x any length.
            03  filler identified by "name".
               05  json-name pic x any length.
            03  filler identified by "surname".
               05  json-surname pic x any isc.
            03  filler identified by "email".
               05  json-email pic x any length.
            03  filler identified by "country".
               05  json-country pic x any length.
```

Let's examine how the processing of a web request is done in an Angular controller:

Take a look at the getNextCustomer method:

```
$scope.getNextCustomer = function() {
     $http.get("servlet/isCobol(AWEBX_NEXT)")
           .then(function (response) {
                if (response.data._comm_buffer._status=="OK")
                    $scope.customer = response.data._comm_buffer;
          })
}
```

It calls the isCOBOL program, using the entry point AWEBX_NEXT. This call is asynchronous, meaning that the javascript code will continue executing while the http object is fetching the data.

When the server returns with data (or an error), the .then() method will be called.

The .then method expects as a parameter a function which receives a response object as its own parameter. The response.data field contains the response model defined in the AWEBX.cbl file.

The function needs to check if the fetch operation was successful:

```
if (response.data._comm_buffer._status=="OK")
```

and, if so, it will extract the customer model and make it available in the controller:

```
$scope.customer = response.data._comm_buffer;
```

This will automatically display the model data in the input tags of the form. Each input has an ng-model directive, which holds the field that will be bound to the edit field:

```
<input type="text" class="form-control" ng-
model="customer.name" id="edFirstname" style="width:280px" />
```

As the user modifies the content of the input field, the model in the controller is automatically updated.

So, all we need to do in order to save the changes is to post the model to the isCOBOL program.

This is done in the saveCustomer (or newCustomer) method of the controller. All we need to do is call the isCOBOL program, using the right entry point, AWEBX_UPDATE (AWEBX_INSERT), and pass it the customer model:

```
$scope.saveCustomer = function() {
            callServerWithJson("AWEBX_UPDATE",
$scope.customer, $scope.onSuccess, $scope.onError);
}
```

The *callServerWithJson* utility method accepts the entry point to call, a model to pass to the isCOBOL program, and 2 callbacks, that specify the function to execute if the http request is successful *onSuccess*, or if it fails onError.

Notice that the *onSuccess* will be called even if the isCOBOL program generates an error (duplicated key, record locked, and so on). This is because the HTTP request was carried out successfully, but a logical program error occurred. So the *onSuccess* method needs to check the response object and handle it appropriately.

The *onError* callback will be invoked only if the http request fails (network error, server error,...).

Chapter 5

COBOL Servlet option (OOP)

Introduction

One of the initial purposes of the Java language was to enable programmers to make Web pages more interactive by embedding programs called applets. When a browser loads a Web page containing an applet, the browser downloads the applet byte code and executes it on the client machine. However, because of client compatibility, bandwidth, security and other issues, businesses needed an alternative solution where Web pages could be made to interact with server-side instead of client-side Java programs.

Server-side Java programming solves problems associated with applets. A servlet can be thought of as a server-side applet. However, when the code is executed on the server-side, there are no issues with browser compatibility or download times. The servlet byte code runs entirely on the server and only sends information to the client in a form that the client can understand.

Similar to a CGI program, a servlet takes requests from a client such as a Web browser, accesses data, applies business logic, and returns the results.

The servlet is loaded and executed by the Web server, and the client communicates with the servlet through the Web server using HTTP requests. This means that if your Web server is behind a firewall, your servlet is secure.

Servlet technology was developed to improve upon and replace CGI programs. Servlet technology is superior to CGI but uses the same HTML code. So you can switch from CGI programs to servlets on the back-end without having to change the programming on the front-end. Servlets use the CGI protocol.

In addition to Java technology's platform independence and promise of write once, run anywhere, servlets have other advantages over CGI programs:

- Servlets are persistent. They are loaded only once by the Web server and can maintain services such as database connections between requests.
- Servlets are fast. They need to be loaded only once by the Web server. They handle concurrent requests on multiple threads rather than in multiple processes. Thus, applications with servlets perform better and are more scalable than the same applications using CGI programs.
- Servlets are platform and Web server independent.
- Servlets can be used with a variety of clients, not just Web browsers.
- Servlets can be used with a variety of client-side and server-side Web programming techniques and languages.

The isCOBOL EIS introduces a new way to develop COBOL programs that acts like java servlet using *HTTPHandler* class functionality.

One of the most remarkable differences between COBOL servlets and CGI programs is that Web servers automatically maintain user session state for servlets. This means that the COBOL servlet can store user-session specific information in a user session object and retrieve that information on a subsequent call.

The isCOBOL EIS Framework uses this feature to associate the user session with a COBOL thread context. This makes sure that the same instances of COBOL programs get used each time they are called during a particular user session. In other words, COBOL programs called during a particular user session retain their file states and working-storage data between requests from that user session. If desired, the programmer can cancel the program at any time with the CANCEL statement. In fact, at first it will be necessary to cancel old CGI programs because they were written to assume that they have been cancelled between calls. Later, the CANCEL statement can be removed as the old CGI programs are updated to make use of the Stateful nature of COBOL servlets.

COBOL Servlet Programming

Following you will find an explanation about how to develop a simple COBOL servlet that builds an HTML page using a header.htm page and a footer.htm page, filling them with the correct message and sending a text string between them, the string is "Hello world from isCOBOL!".

The Web Servlet container used for this example is Tomcat 7.

This program needs to take the following steps:

- Create a folder called doctest with the following structure:

```
doctest/
        WEB-INF/
                classes
                lib
```

- Create a file called *web.xml* in doctest/WEB-INF folder with the following content:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://
java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://
java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>isCOBOL EIS</display-name>
  <welcome-file-list>
    <welcome-file>Hello.htm</welcome-file>
  </welcome-file-list>
  <filter>
        <filter-name>isCOBOL filter</filter-name>
        <filter-class>com.iscobol.web.IscobolFilter</filter-class>
  </filter>
  <filter-mapping>
        <filter-name>isCOBOL filter</filter-name>
        <url-pattern>/servlet/*</url-pattern>
  </filter-mapping>
  <servlet>
        <servlet-name>isCobol</servlet-name>
        <servlet-class>com.iscobol.web.IscobolServletCall</servlet-class>
  </servlet>
  <servlet-mapping>
        <servlet-name>isCobol</servlet-name>
        <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
  <listener>
   <listener-class>com.iscobol.web.IscobolSessionListener</listener-class>
  </listener>
</web-app>
```

- Create a *Hello.htm* web form to call a COBOL servlet called HELLO:

```
<HTML><HEAD><TITLE>isCOBOL Example</TITLE></HEAD>
<BODY>
<H2>isCOBOL Example.</H2>
<H3>This example shows how easily you can compose an HTML page with an isCOBOL program
running on the web server. The HTML page is composed of two parts; a header and a
footer. The isCOBOL program will insert a message to the header and footer, as well as
inserting the text "Hello world from isCOBOL" between the header and footer.</H3>
<HR size="2">
<FORM method="post" action="servlet/isCobol(HELLO)">
    <p><input type="submit" value="Invoke isCOBOL HELLO program" /></p>
</FORM>
```

Note that in POST method of HTML form there is the call of the COBOL Servlet called HELLO.

- Create a *Header.htm* as follow:

```
<HTML>
<HEAD><TITLE>CGI Header</TITLE></HEAD>
<BODY>
<CENTER>
<H1>This is the header HTML page of the isCOBOL Example</H1>
<H2>This is the message sent by the isCOBOL program: %%opening-message%%</H2>
<HR>
```

Note that this form displays the top of the HTML page that the program HELLO.cbl will build; as we can see, the <HTML>, <BODY> and <CENTER> tags are not closed, and there is the string %%opening-message%% that will be managed and replaced by the COBOL servlet program.

- Create a *Footer.htm* web form as follow:

```
</CENTER>
<BR>
<HR>
This is the footer HTML page of the isCOBOL Example.
<H2>This is the message sent by the isCOBOL program: %%closing-message%%</H2>
</BODY></HTML>
```

Note that this form displays the bottom of the HTML page that the program HELLO.cbl will build. Here the tags <HTML>, <BODY> and <CENTER> are closed and there is the string %%closing-message%% that will be managed and replaced by the COBOL servlet program.

- Create a *HELLO.cbl* COBOL Servlet program as follows:

```
       PROGRAM-ID. HELLO initial.
       CONFIGURATION SECTION.
       REPOSITORY.
       class web-area as "com.iscobol.rts.HTTPHandler"
           .
       WORKING-STORAGE SECTION.
       01  hello-buffer pic x(40) value "Hello World from isCOBOL!".
       01  rc pic 9.
       01  html-header-form identified by "Header".
            05  identified by "opening-message".
                10 opening-message pic x(40).
       01  html-footer-form identified by "Footer".
            05  identified by "closing-message".
                10  closing-message pic x(40).

       LINKAGE SECTION.
       01 comm-area object reference web-area.
       PROCEDURE DIVISION using comm-area.
       MAIN-LOGIC.
           move "This is the header" to opening-message
           set rc = comm-area:>processHtmlFile (html-header-form).
           comm-area:>displayText  (hello-buffer).
           move "Bye Bye by isCOBOL" to closing-message
           set rc = comm-area:>processHtmlFile (html-footer-form).
           goback.
```

Note that the COBOL servlet does the following steps:

- Move the value "This is the header" to the variable opening-message of the structure prepared for Header.htm
- Add to the HTML page source (that currently is empty) the Header.htm form replacing the string %%opening-message%% by the opening-message variable value
- Add to the HTML page the text "Hello world from isCOBOL!"
- Move the value "Bye Bye by isCOBOL" to the variable closing-message of the structure prepared for Footer.htm
- Add to the HTML page source the Footer.htm form replacing the string %%closing-message%% by the closing-message variable value

at the exit of the program, the page HTML will be sent to the Web Server.

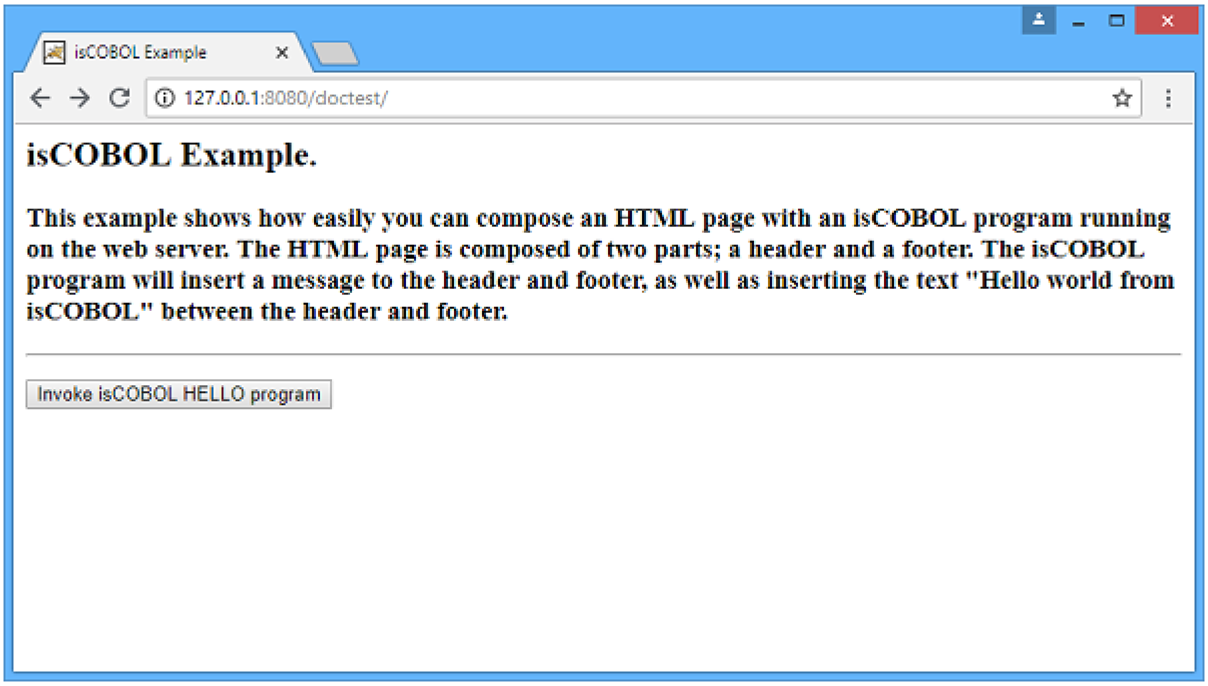
- Compile HELLO.cbl without any specific options and copy HELLO.class under doctest/WEB-IF/classes folder
- Create a iscobol.properties file under doctest/WEB-IF/classes folder with a property to inform isCOBOL EIS framework of the path of all HTML useful files:

```
iscobol.http.html_template_prefix=webapps/doctest
```

- Copy the isCOBOL runtime library (iscobol.jar) to the doctest/WEB-IF/lib folder.
- Create a war file to be deployed in Tomcat called doctest.war that includes all files of doctest folder. It can be done with the following command:

```
jar -cfv doctest.war *
```

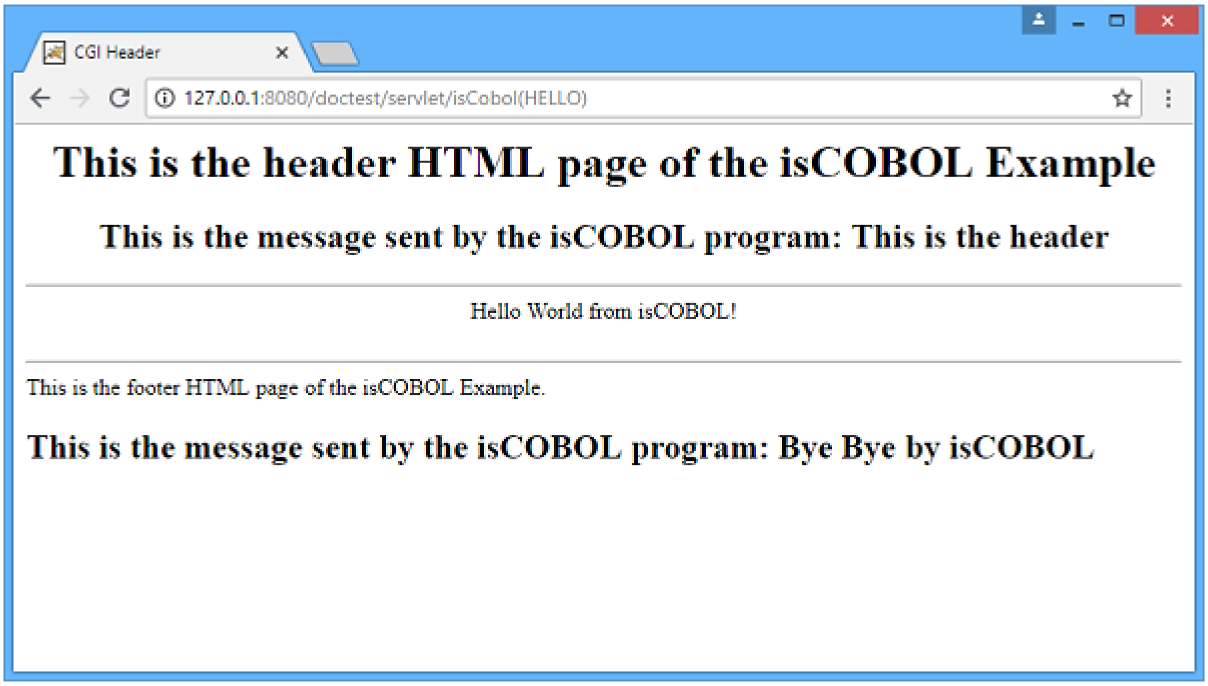
Once doctest.war file is deployed correctly in Tomcat servlet container, we can try it using http://127.0.0.1:8080/doctest, assuming Tomcat is running on the localhost and using the default port, 8080.



isCOBOL Example.

This example shows how easily you can compose an HTML page with an isCOBOL program running on the web server. The HTML page is composed of two parts; a header and a footer. The isCOBOL program will insert a message to the header and footer, as well as inserting the text "Hello world from isCOBOL" between the header and footer.

By pressing the "Invoke isCOBOL Hello program" button, the result is:



This is the header HTML page of the isCOBOL Example

This is the message sent by the isCOBOL program: This is the header

Hello World from isCOBOL!

This is the footer HTML page of the isCOBOL Example.

This is the message sent by the isCOBOL program: Bye Bye by isCOBOL

COBOL Servlet Programming with AJAX and XML

AJAX (Asynchronous JavaScript and XML) is a group of interrelated Web development techniques used on the client side to create asynchronous Web Applications. With Ajax, Web applications can send data to, and retrieve data from, a server asynchronously (in the background) without interfering with the display and behavior of the existing page.

Extensible Markup Language (XML) is a text format derived from Standard Generalized Markup Language (SGML). Compared to SGML, XML is simple. HyperText Markup Language (HTML), by comparison, is even simpler. Even so, a good reference book on HTML is an inch thick. This is because the formatting and structuring of documents is a complicated business.

Most of the excitement about XML is related to a new role as an interchangeable data serialization format. XML provides two enormous advantages as a data representation language:

- It is text-based
- It is position-independent

The scope of this paragraph is to show how to develop a simple web application that uses XML stream to communicate data from COBOL servlet to a Web form.

The following example called HELLO.cbl, is located in sample/eis/http/xml folder. The README.txt file explains how it works and how to deploy it.

This example needs to take the following steps:

- Create an HTML file called index.html that is able to establish an AJAX communication to receive an XML stream from COBOL servlet program:

  In *index.html* there is included a Javascript code based on JQUERY to be able to call some COBOL servlet entry points making a GET request type (default) and receiving XML data stream:

```
   function callServer (cobolProg) {
      var url = "servlet/isCobol(" + cobolProg + ")";
      jQuery.ajax(url, {
                    success: handleSuccess,
                    error: handleError
                 });
      return false;
   }
```

- Load all COBOL Servlets using the following statement:

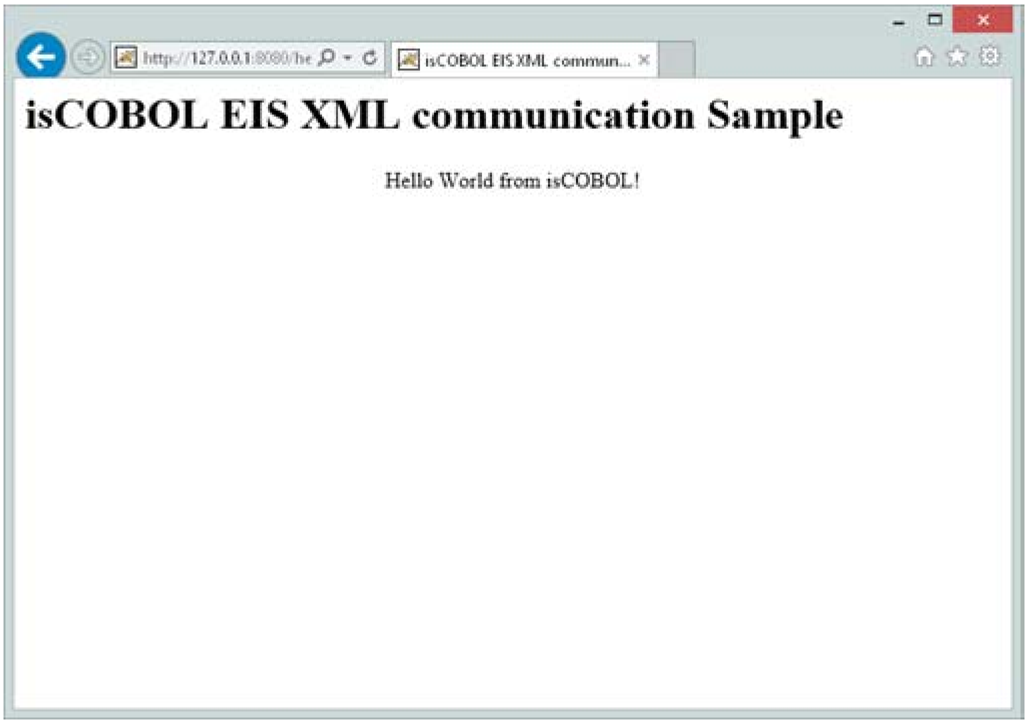
```
window.onload = callServer('HELLO');
```

  Note that when HELLO COBOL Servlet is loaded the following code executed:

```
      move "Hello World from isCOBOL!" to xml-hellotext.
      lnk-area:>displayXML (hello-buffer).
```

  And XML stream is returned to the Web form with the displayXML() command.

When running this example the result is the following:



isCOBOL EIS XML communication Sample

Hello World from isCOBOL!

COBOL Servlet Programming with AJAX and JSON

JSON or JavaScript Object Notation, is an open standard format that uses text easy to understand to transmit data objects consisting of attribute-value pairs. It is used primarily to transmit data between a server and web application, as an alternative to XML.

Although originally derived from the JavaScript scripting language, JSON is an independent data format, and code for parsing and generating JSON data is readily available in a large variety of programming languages.

Here we will show how to develop a simple web application of data file management that uses JSON stream to communicate data from COBOL servlet to HTML pages.

The following example is located in sample/eis/http/json folder. The README.txt file explains how it works and how to deploy it.

This example needs to take the following steps:

- Create an HTML file that is able to establish an AJAX communication using JSON stream to a COBOL servlet program:

In *awebx.htm* there is included a Javascript code based on JQUERY to be able to call some COBOL servlet entry point making a GET request type (default) and receiving JSON data stream:

```
function callServer (cobolProg) {
    var url = "servlet/isCobol(" + cobolProg + ")";
    var parm = $("form").serialize();
      $.ajax(url, {
                    success: handleSuccess,
                    error: handleError,
                    data: parm
                 });
      return false;
   }
```

- Load all COBOL Servlet entry points using the following statement:

```
callServer("AWEBX"); // program initialization
```

Note the once AWEBX COBOL Servlet is loaded the INIT paragraph is executed:

```
       INIT.
           set declaratives-off to true.
           move low-values to r-awebx-email.
           open i-o awebxfile.
           set declaratives-on to true.
           if file-status > "0z" and file-status not = "41"
              open output awebxfile
              close awebxfile
              open i-o awebxfile.
           comm-area:>displayJSON (ok-page).
           goback.
```

Code is included that associates each AWEBX entry point to an HTML button, to be executed when the button is clicked:

```
<input type="submit" value="Insert" onclick="return callServer('AWEBX_INSERT');">
<input type="submit" value="Search" onclick="return callServer('AWEBX_SEARCH');">
<input type="submit" value="Next"   onclick="return callServer('AWEBX_NEXT');">
<input type="submit" value="Update" onclick="return callServer('AWEBX_UPDATE');">
<input type="submit" value="Delete" onclick="return callServer('AWEBX_DELETE');">
```

Note that the above HTML is able to call the following COBOL servlet entry-point:

```
       INSERT-VALUES.
           entry "AWEBX_INSERT" using comm-area.
           ...
           goback.

       SEARCH-VALUES.
           entry "AWEBX_SEARCH" using comm-area.
           ...
           goback.

       NEXT-VALUES.
           entry "AWEBX_NEXT" using comm-area.
           ...
           goback.

       UPDATE-VALUES.
           entry "AWEBX_UPDATE" using comm-area.
           ...
           goback.
```

- Define some fields in HTML to input data suitable for data management, such as name, surname, email, country etc:

```
     <input type="text" name="name" size="25" placeholder="Name"/><br/>
     <input type="text" name="surname" size="25" placeholder="Surname"/><br/>
     <input type="text" name="email" size="25" placeholder="E-mail"/><br/>
     <select name="country"  placeholder="Country">
      <option value="" selected="selected" disabled="disabled">Country</option>
       <option value="us">US</option>
       <option value="it">Italy</option>
       <option value="fi">Finland</option>
       <option value="nl">The Netherlands</option>
       <option value="de">Germany</option>
       <option value="fr">France</option>
       <option value="sp">Spain</option>
       <option value="uk">United Kingdom</option>
     </select><br/>
```

- In COBOL Servlet create a working storage structure that matches the field name of previous HTML. It can be done with identified by clause:

```
      01 comm-buffer identified by "_comm_buffer".
         03  filler identified by "_status".
             05 response-status pic x(2).
         03  filler identified by "_message".
             05 response-message pic x any length.
         03  filler identified by "name".
             05  json-name pic x any length.
         03  filler identified by "surname".
             05  json-surname pic x any length.
         03  filler identified by "email".
             05  json-email pic x any length.
         03  filler identified by "country".
             05  json-country pic x any length.
```

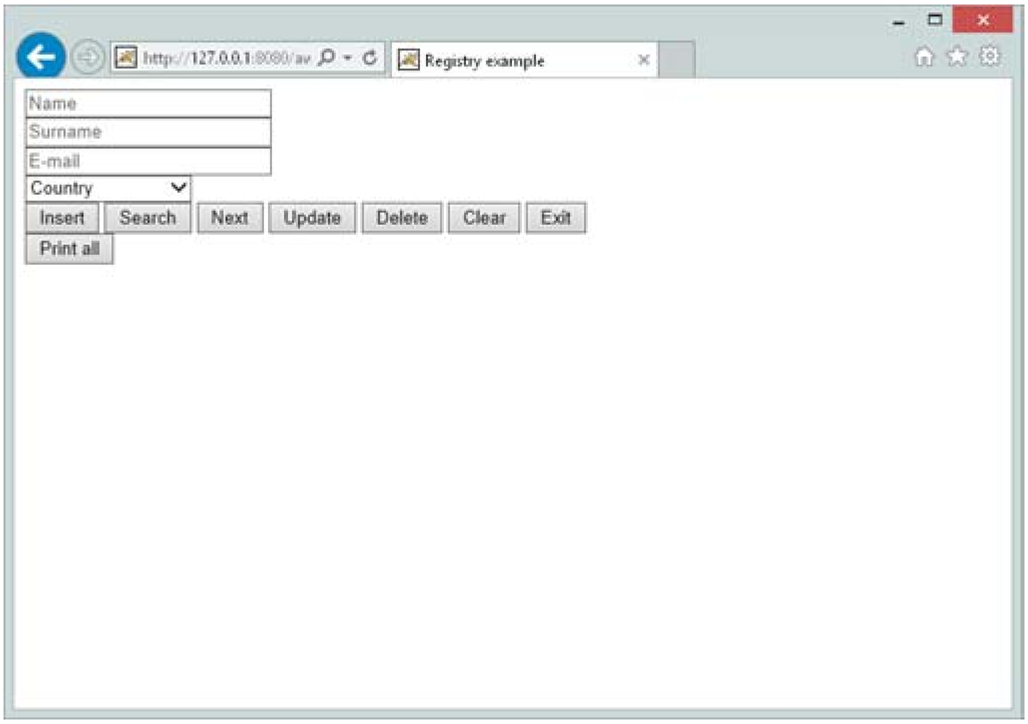
- In COBOL Servlet manage GET request by accept() answering with a JSON stream by displayJSON(). For example if "insert" is submitted the following entry point is invoked:

```
       INSERT-VALUES.
           entry "AWEBX_INSERT" using comm-area.
           comm-area:>accept(comm-buffer).
           move spaces to error-status.
           perform check-values.
           if error-status = spaces
              move json-name    to r-awebx-name
              move json-surname to r-awebx-surname
              move json-email   to r-awebx-email
              move json-country to r-awebx-country
              write rec-awebxfile
              move "Operation successful" to ok-message;;
              comm-area:>displayJSON (ok-page)
           else
              comm-area:>displayJSON (error-page).
           goback.
```

- In a similar way when a "next" command is submitted, the records are returned back as JSON stream with the displayJSON() command:

```
       NEXT-VALUES.
           entry "AWEBX_NEXT" using comm-area.
           read awebxfile next
           move r-awebx-name    to json-name
           move r-awebx-surname to json-surname
           move r-awebx-email   to json-email
           move r-awebx-country to json-country
           move "OK" to response-status
           move "" to response-message;;
           comm-area:>displayJSON (comm-buffer).
           goback.
```

This is the output form of awebx.htm used in this example:



COBOL Servlet Programming to replace CGI COBOL programming

This page will show you how to migrate older CGI COBOL programs to isCOBOL Servlets in order to take advantage of useful features of the HTTPHandler class. Usually only a few changes are required; most of your source code will remain unchanged.

Conversion of the ACUCOBOL-GT Oscars sample

The following example is located in sample/eis/http/getpost/acucgi2is folder. The README.txt file explains how it works and how to deploy it.

This example needs to take the following steps:

- A POST form action invoking the COBOL program acting as a servlet to replace the CGI program:

```
<FORM method="post" action="servlet/isCobol(OSCARS)">
```

This HTML document, called 'oscars.htm' will also include these controls:

```
<input type=checkbox name=y1996 value=1996> 1996
<input type=checkbox name=y1995 value=1995> 1995
<input type=checkbox name=y1994 value=1994> 1994
<input type=checkbox name=y1993 value=1993> 1993
<P>
<input type=checkbox name=y1992 value=1992> 1992
<input type=checkbox name=y1991 value=1991> 1991
<input type=checkbox name=y1990 value=1990> 1990
<input type=checkbox name=y1989 value=1989> 1989
<P>
<input type=checkbox name=y1988 value=1988> 1988
<input type=checkbox name=y1987 value=1987> 1987
<input type=checkbox name=y1986 value=1986> 1986
<input type=checkbox name=y1985 value=1985> 1985
<P>
<input type="submit" value="Submit Query" >
```

When checking one or more years and pressing the 'Submit query' button, the OSCARS COBOL servlet program is called.

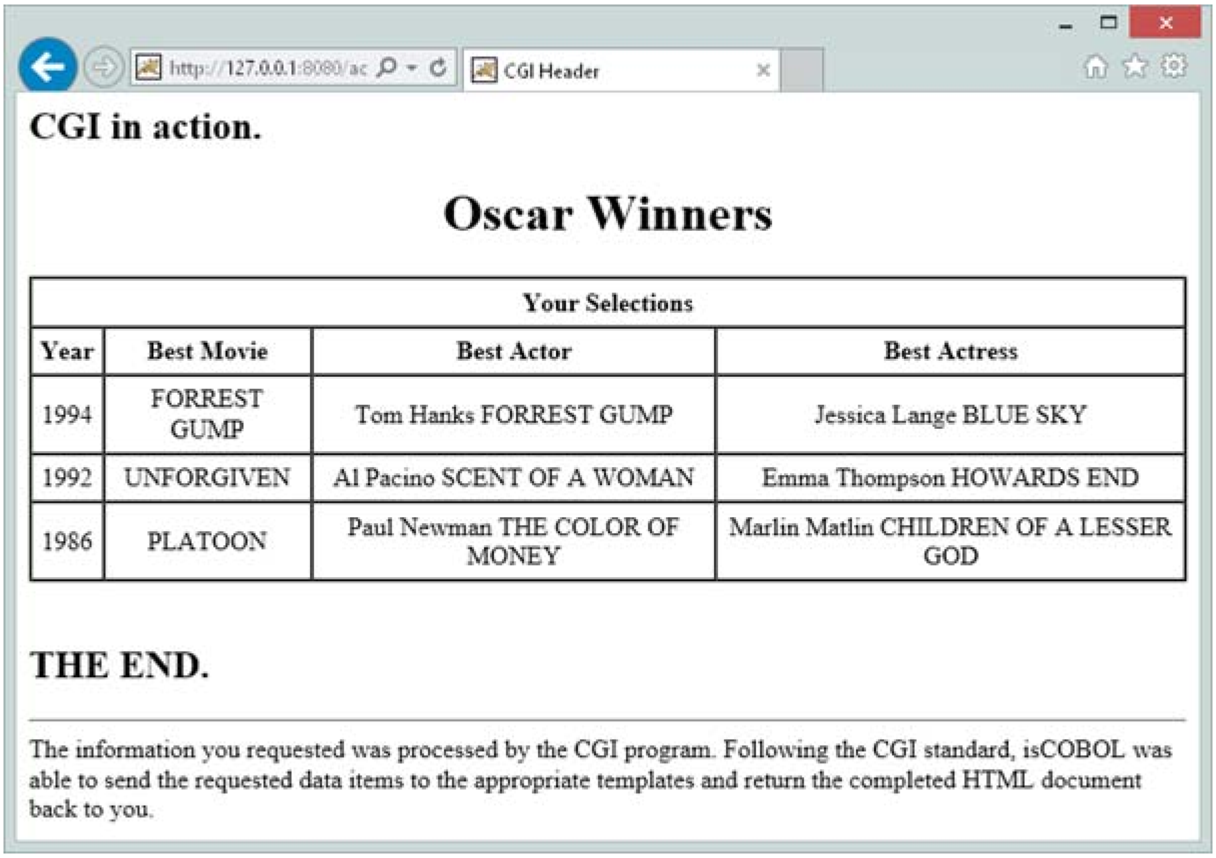
The OSCARS COBOL program needs to be compiled with the following options:

```
-ca -smat
```

The following setting is required in the Runtime configuration:

```
iscobol.http.stateless=true
```

The result choosing 1994, 1992 and 1986 is the following:



CGI in action.

Oscar Winners

Your Selections			
Year	Best Movie	Best Actor	Best Actress
1994	FORREST GUMP	Tom Hanks FORREST GUMP	Jessica Lange BLUE SKY
1992	UNFORGIVEN	Al Pacino SCENT OF A WOMAN	Emma Thompson HOWARDS END
1986	PLATOON	Paul Newman THE COLOR OF MONEY	Marlin Matlin CHILDREN OF A LESSER GOD

THE END.

The information you requested was processed by the CGI program. Following the CGI standard, isCOBOL was able to send the requested data items to the appropriate templates and return the completed HTML document back to you.

Conversion of the Micro Focus sample

Using the above approach is also possible to migrate a Micro Focus COBOL CGI program to COBOL Servlet.

Under sample/eis/http/getpost/ mfcgi2is folder you find an example of a Micro Focus Cobol CGI program rewritten to run with the HTTP option of isCOBOL EIS.

The README.txt file explains how it works and how to deploy it.

This example needs to take the following steps:

- A POST form action invoking the COBOL program acting as a servlet to replace the CGI program:

```
<BODY><FORM id=form1 name=form1 action="servlet/isCobol(WEBDEMO)" method=post >
```

This HTML document, called 'WebDemo.htm', will also include these controls:

```
<INPUT id=checkbox1 type=checkbox value=on name=checkbox1>Vanilla
<INPUT id=checkbox2 type=checkbox value=on name=checkbox2>Chocolate
<INPUT id=checkbox3 type=checkbox value=on name=checkbox3>Marble

<INPUT id=radiobutton1 type=radio value=White name=radio>White
<INPUT id=radiobutton2 type=radio value=Chocolate name=radio>Chocolate
<INPUT id=radiobutton3 type=radio value=Blue name=radio>Blue

<SELECT id=select1 name=select1> <OPTION
selected>Cash<OPTION>Visa<OPTION>Check<OPTION>Mac</OPTION></SELECT>
```

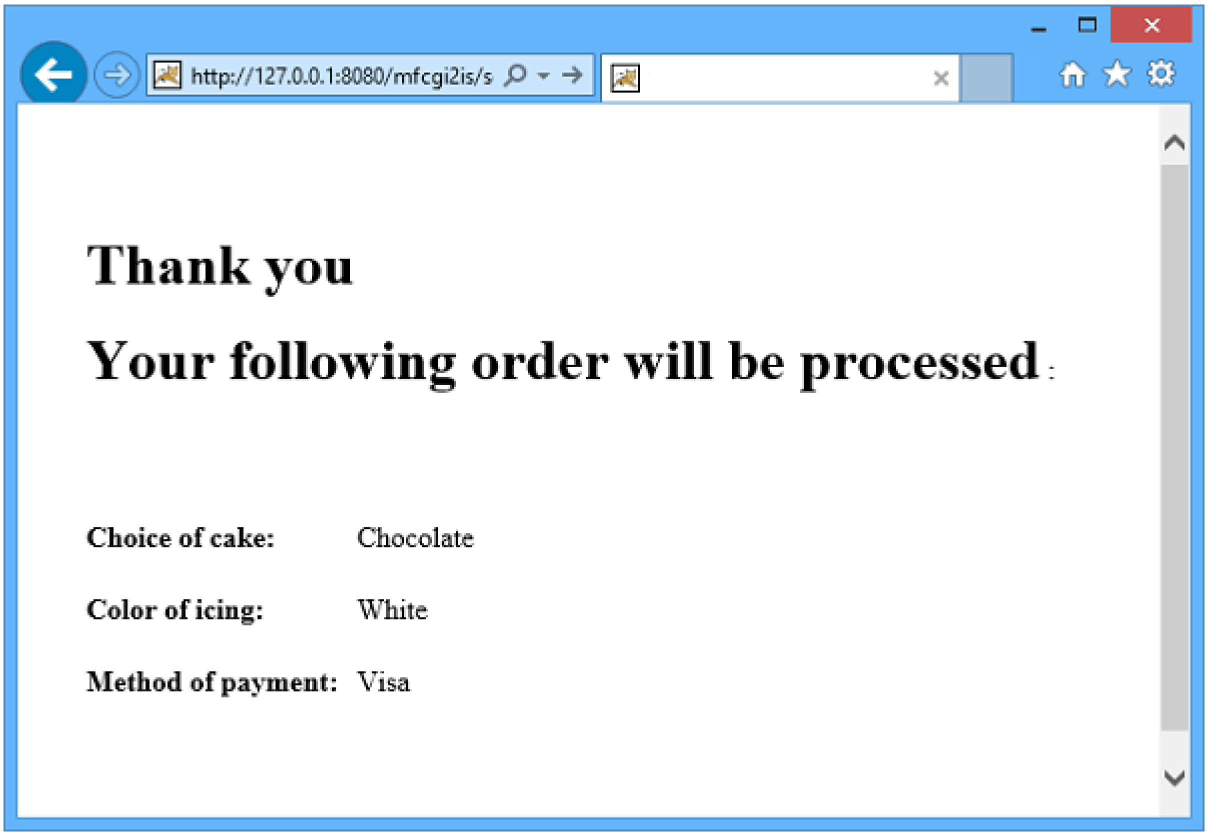
The WEBDEMO COBOL program needs to be compiled with the following options:

```
-sa -exec=html
```

having the following regexp in the Compiler configuration:

```
iscobol.compiler.regexp="(?i)(STOP)\\s+(RUN)" "GOBACK" \
                       "(?i)(local-storage)\\s+(section.)" "|local-storage section." \
                        "call-convention" "|call-convention" \
                        "(?i)(\\(length )" "(length of " \
                        "(?i)( length )" "length of " \
                        "(?i)(CALL)\\s+(LLNK)" "CALL"
```

The result choosing Chocolate cake, White icing and Visa payment is the following:


http://127.0.0.1:8080/mfcgi2is/s

Thank you

Your following order will be processed :

Choice of cake:	Chocolate
Color of icing:	White
Method of payment:	Visa

Chapter 6

WebDirect option

Introduction

With isCOBOL EIS WebDirect option your organization can leverage existing COBOL syntax to develop and deploy a universally accessible, zero client, rich Internet applications (RIA) using standard COBOL screen sections and existing program procedure flow. No knowledge of object-oriented programming, JavaScript, HTML, or other Web languages is required.

WebDirect is a Java framework for presenting a graphical user interface, composed of elements such as windows, dialogs, menus, text fields and buttons, inside a Web Browser. This technology uses AJAX (asynchronous JavaScript and XML) techniques and the Comet web application model. The web application is deployed as a Servlet and therefore requires a Java-enabled web server, one that implements the Java Servlet specification from Sun Microsystems

WebDirect takes advantage of ZK libraries, installed with the product. ZK is an event-driven, component-based framework to enable rich user interfaces for Web applications. ZK includes an AJAX-based event-driven engine, a rich component set of XUL and XHTML and a markup language called ZUML (ZK User Interface Markup Language).

The current version of ZK is:

- ZK 8.5

Technical Notes

isCOBOL EIS WebDirect on the client side is a JavaScript application running inside a web browser. This environment has many limitations in comparison with a full GUI environment, e.g. only a few events are generated. JavaScript is a script language so its performance is not as good as compiled languages, although latest generation browsers are improving performance by the use of JIT (Just In Time) compilers.

WebDirect was not developed from scratch; it uses a library, ZK, that hides the JavaScript implementation and exposes a Java API. Veryant interfaces our set of GUI controls with the ones implemented in ZK. As a result, because our controls are similar to those provided by ZK, future updates will require less effort and provide more stable releases interfacing with ZK GUI controls. Alternatively, controls completely different from the ZK controls will require more development and testing time.

The client/server communication is performed through the HTTP protocol; since this protocol is very limited in functionality, a special technique (called "COMET") has been used in order to get the needed functionality. This technology is the up-to-date best technology in this area. However its performance is not as good as native protocols. Just as an example, it uses XML protocols, so it creates bigger messages and it requires considerable computation resources for marshalling.

Installation Environment

In order to deploy and run programs using WebDirect, the programs' environment must be set up in a servlet container such as Apache Tomcat.

Veryant recommends using Apache Software Foundation Tomcat version 7 or higher for running WebDirect applications.

The Apache Tomcat main page is http://tomcat.apache.org/

WebDirect is expected to work also on the following containers:

- IBM WebSphere
- BEA WebLogic
- Oracle OC4J and Oracle OPMN Release 3
- Liferay
- Pluto
- Jetty
- Resin
- WildFly (ex JBoss AS)

Servlet Container and Web Browser Requirements

WebDirect runs on any web server that supports Servlet 2.3+ and JVM 1.5+.

The web browser must be able to run JavaScript and support Ajax (namely the XMLHttpRequest object). The following browsers are certified for ZK 8:

- Internet Explorer 8+
- Edge
- Firefox
- Chrome
- Safari
- iOS Safari
- Android Browser
- Opera

Getting Started

The jar libraries must be copied into the proper directory in order to be available to the web application. If you're using Tomcat, you must copy these libraries in the "lib" folder of your web application.

WebDirect is composed of:

Name	Description	Location
iscobol.css	WebDirect stylesheet	resources/css
iscobol.properties	Configuration file for the web application	WEB-INF/classes
iscobol.jar	isCOBOL Runtime Framework	WEB-INF/lib

Name	Description	Location
iswd2.jar	isCOBOL WebDirect Implementation	WEB-INF/lib
javassist.jar itext-2.1.7v5.jar xmlbeans-3.1.0.jar poi-4.1.2.jar poi-ooxml-4.1.2.jar poi-ooxml-schemas-4.1.2.jar	Additional isCOBOL libraries	WEB-INF/lib
bsh.jar commons-codec.jar commons-collections.jar commons-fileupload.jar commons-io.jar commons-logging.jar Filters.jar flashchart.jar gmapsz.jar gson.jar jackson-annotations.jar jackson-core.jar jackson-databind.jar sapphire.jar silvertail.jar slf4j-api.jar slf4j-jdk14.jar timelinez.jar timeplotz.jar zcommon.jar zel.jar zhtml.jar zk-bootstrap.jar zk.jar zkbind.jar zkex.jar zkmax.jar zkplus.jar zml.jar zsoup.jar zul.jar zweb.jar	ZK Framework and its dependences	WEB-INF/lib
portlet.xml	ZK loader for ZUML pages	WEB-INF
web.xml	Deployment Descriptor. To configure servlets, listeners and an optional filter	WEB-INF
zk.xml	Configuration descriptor of ZK. This file is optional. If you need to configure ZK differently from the default, you could provide a file called zk.xml under the WEB-INF directory.	WEB-INF

All the above files are installed in $ISCOBOL/eis/webdirect.

Running the sample application

WebDirect comes with a sample web application. This chapter explains how to deploy and run the sample application.

1. Build the war

1. Change to the webdirect folder of isCOBOL samples
   Windows

```
cd %ISCOBOL%\sample\eis\webdirect\widget
```

   Linux/Unix

```
cd $ISCOBOL/sample/eis/webdirect/widget
```

2. Add zk.jar, zul.jar and zcommons.jar to the CLASSPATH
   Windows

```
set
CLASSPATH=%CLASSPATH%;..\..\..\..\eis\webdirect\lib\zk.jar;..\..\..\..\eis\webdirect\l
ib\zul.jar;..\..\..\..\eis\webdirect\lib\zcommon.jar
```

   Linux/Unix

```
export CLASSPATH=$CLASSPATH:../../../../eis/webdirect/lib/zk.jar:../../../../eis/
webdirect/lib/zul.jar:../../../../eis/webdirect/lib/zcommon.jar
```

3. Compile the programs
   Windows

```
iscc -sp=../../../isdef;copylib -wd2 *.cbl
```

   Linux/Unix

```
iscc -sp=../../../isdef:copylib -wd2 *.cbl
```

4. Create the "wd2" webapp folder structure as follows:
   Windows

```
mkdir webdirect
mkdir webdirect\arc
mkdir webdirect\excel
mkdir webdirect\pdf
mkdir webdirect\upload
mkdir webdirect\resources
mkdir webdirect\resources\css
mkdir webdirect\WEB-INF
mkdir webdirect\WEB-INF\classes
mkdir webdirect\WEB-INF\lib
mkdir webdirect\WEB-INF\programs
```

Linux/Unix

```
mkdir webdirect
mkdir webdirect/arc
mkdir webdirect/excel
mkdir webdirect/pdf
mkdir webdirect/upload
mkdir webdirect/resources
mkdir webdirect/resources/css
mkdir webdirect/WEB-INF
mkdir webdirect/WEB-INF/classes
mkdir webdirect/WEB-INF/lib
mkdir webdirect/WEB-INF/programs
```

5. Copy the compiled programs and the sample files to the webapp folder as follows:
   Windows

```
copy %ISCOBOL%\sample\eis\webdirect\widget\css\custom.css webdirect\resources\css
copy %ISCOBOL%\sample\eis\webdirect\widget\images\* webdirect\WEB-INF\programs
copy %ISCOBOL%\sample\eis\webdirect\widget\snippet\* webdirect\arc
copy %ISCOBOL%\sample\eis\webdirect\widget\index.html webdirect
copy %ISCOBOL%\sample\eis\webdirect\widget\iscobol.properties webdirect\WEB-
INF\classes
copy %ISCOBOL%\sample\eis\webdirect\widget\*.class webdirect\WEB-INF\programs
```

   Linux/Unix

```
cp $ISCOBOL/sample/eis/webdirect/widget/css/custom.css webdirect/resources/css
cp $ISCOBOL/sample/eis/webdirect/widget/images/* webdirect/WEB-INF/programs
cp $ISCOBOL/sample/eis/webdirect/widget/snippet/* webdirect/arc
cp $ISCOBOL/sample/eis/webdirect/widget/index.html webdirect
cp $ISCOBOL/sample/eis/webdirect/widget/iscobol.properties webdirect/WEB-INF/classes
cp $ISCOBOL/sample/eis/webdirect/widget/*.class webdirect/WEB-INF/programs
```

6. Copy the isCOBOL runtime and WebDirect libraries to the webapp lib folder as follows:
   Windows

```
copy %ISCOBOL%\eis\webdirect\lib\*.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\commons-logging.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\commons-codec-*.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\javassist.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\iscobol.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\poi-*.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\xmlbeans-*.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\eis\wd2\lib\iswd2.jar webdirect\WEB-INF\lib
copy %ISCOBOL%\lib\itext-*.jar webdirect\WEB-INF\lib
```

Linux/Unix

```
cp $ISCOBOL/eis/webdirect/lib/*.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/commons-logging.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/commons-codec-*.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/javassist.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/iscobol.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/poi-*.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/xmlbeans-*.jar webdirect/WEB-INF/lib
cp $ISCOBOL/eis/wd2/lib/iswd2.jar webdirect/WEB-INF/lib
cp $ISCOBOL/lib/itext-*.jar webdirect/WEB-INF/lib
```

7. Copy deployment descriptors and the standard css file from the isCOBOL distribution to the webapp folders as follows:
   Windows

```
copy %ISCOBOL%\eis\webdirect\css\iscobol.css webdirect\resources\css
copy %ISCOBOL%\eis\webdirect\xml\*.xml webdirect\WEB-INF
```

Linux/Unix

```
cp $ISCOBOL/eis/webdirect/css/iscobol.css webdirect/resources/css
cp $ISCOBOL/eis/webdirect/xml/*.xml webdirect/WEB-INF
```

8. Create the "wd2.war" with the following commands:

```
cd webdirect
jar -cf webdirect.war *
```

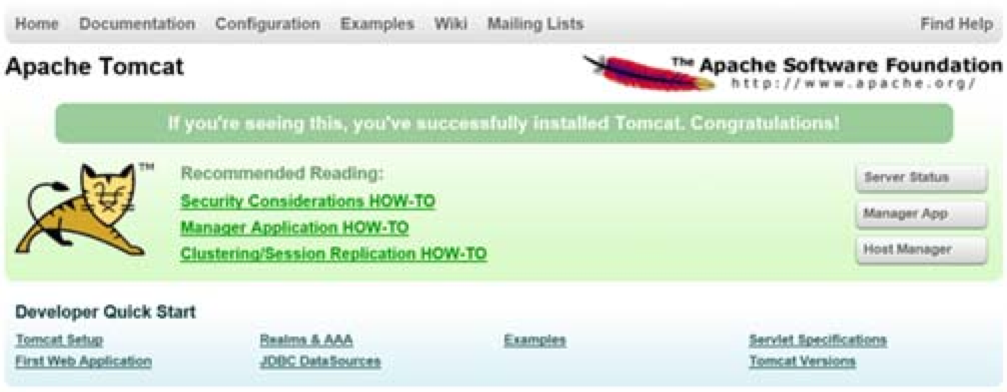
2. Deploy the war

The following instructions are applicable to Apache Tomcat. However, your webapp can also be executed by other servlet containers.

Download Tomcat from http://tomcat.apache.org/ and install it, if you haven't installed it yet. Start the Tomcat service.

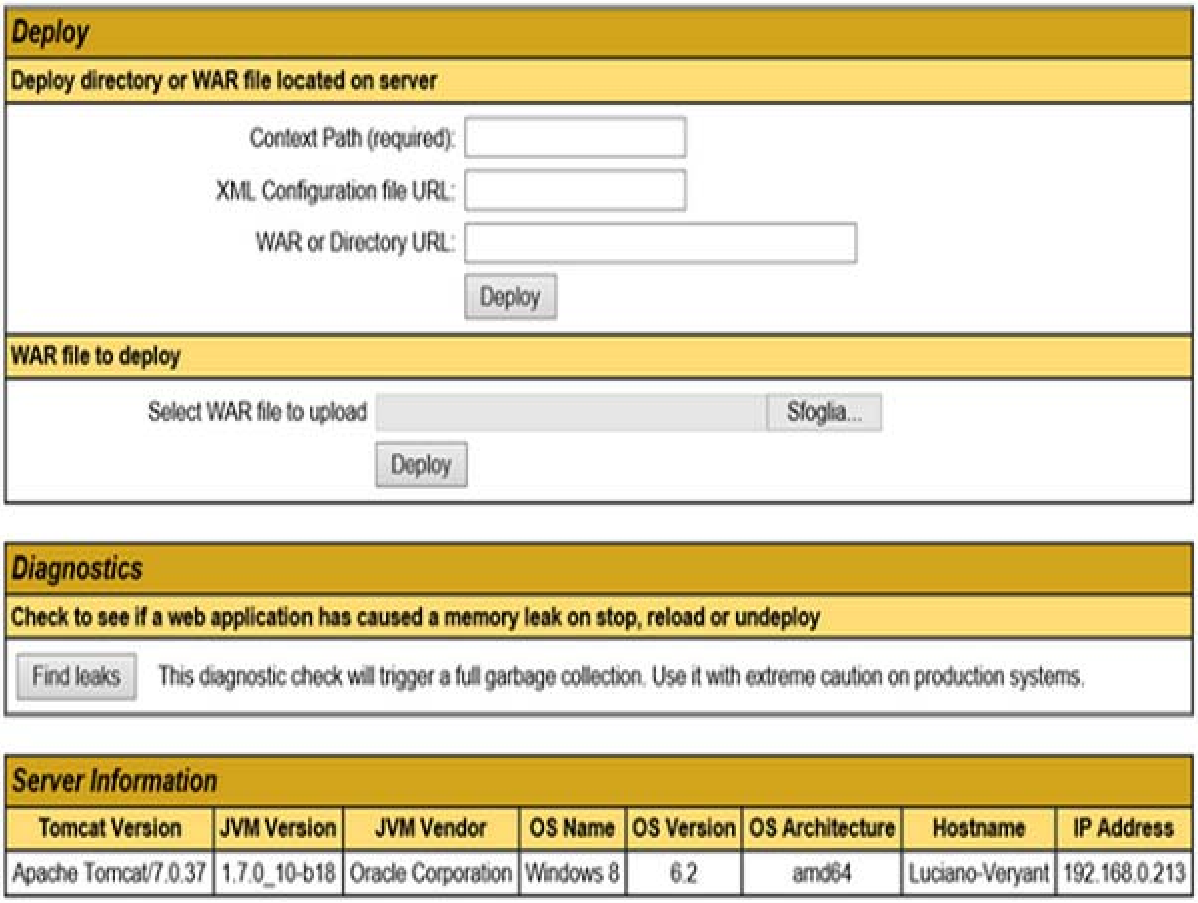
Note: if you're running Tomcat on Unix/Linux, ensure that the working directory is the Tomcat home directory. If you start the process from another directory (e.g. the Tomcat bin directory), then relative paths in the sample will not work.

When Tomcat service is started, open a browser and navigate to "http://127.0.0.1:8080/" . The browser displays something like:



Select *Tomcat Manager* link in order to application administration pages. You will be prompted for username and password. By default Tomcat has the user "admin" with no password. You can refer to *tomcat-users.xml.*

Using the Tomcat Web Application Manager, scroll down to the Deploy dialog and use the *Browse* button to select the Web Application Archive file (webdirect.war)


Deploy
Deploy directory or WAR file located on server
Context Path (required):
XML Configuration file URL:
WAR or Directory URL:
Deploy
WAR file to deploy
Select WAR file to upload
Sfoglia...
Deploy
Diagnostics
Check to see if a web application has caused a memory leak on stop, reload or undeploy
Find leaks
This diagnostic check will trigger a full garbage collection. Use it with extreme caution on production systems.
Server Information

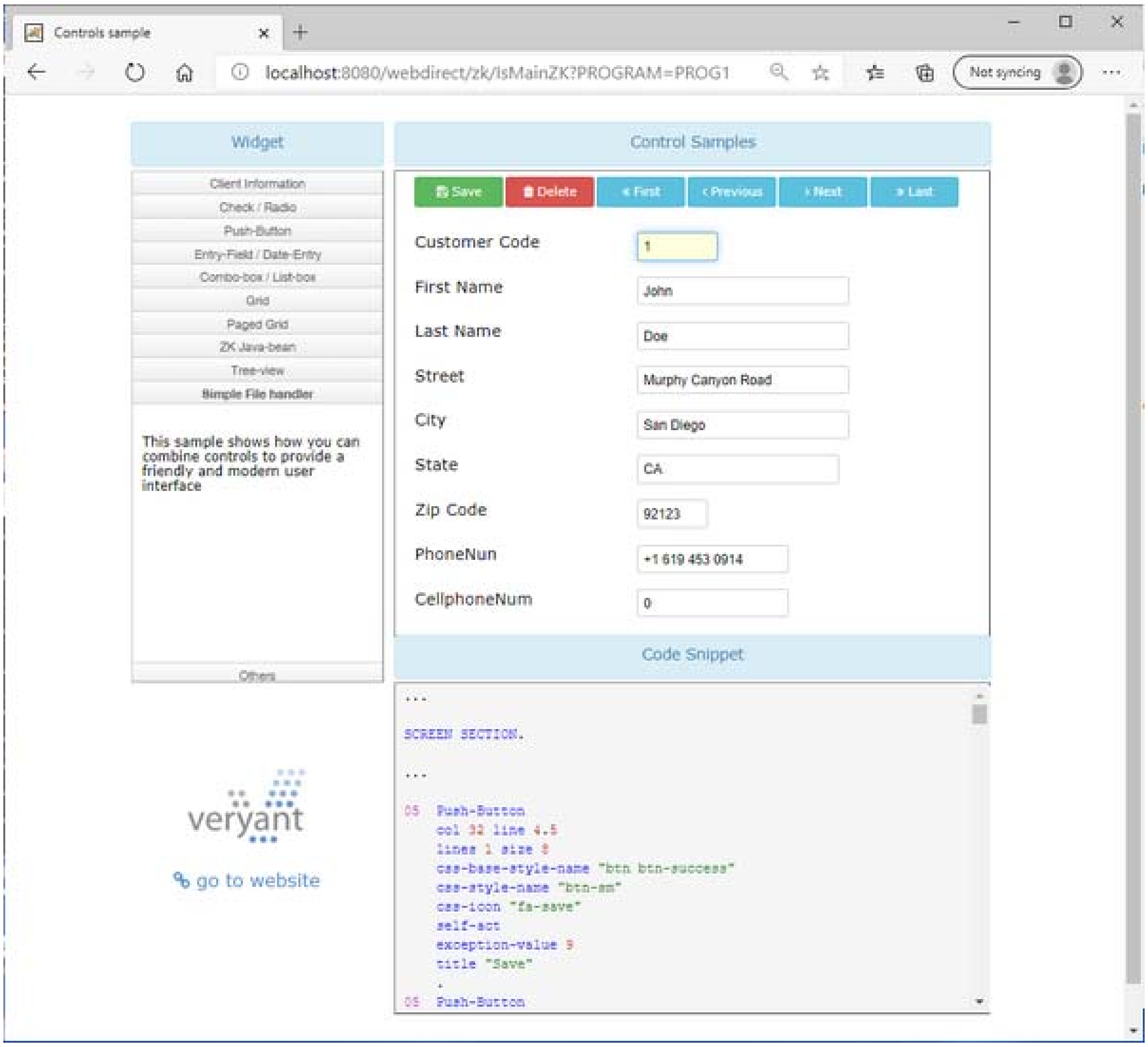
Tomcat Version	JVM Version	JVM Vendor	OS Name	OS Version	OS Architecture	Hostname	IP Address
Apache Tomcat/7.0.37	1.7.0_10-b18	Oracle Corporation	Windows 8	6.2	amd64	Luciano-Veryant	192.168.0.213

An item called "webdirect" will be added to the Applications list.

Edit the file iscobol.properties in classes' folder to insert valid license codes. The following licenses are required:

- isCOBOL Runtime license (iscobol.license.2021)
- isCOBOL EIS license (iscobol.eis.license.2021)

To run the sample, open a browser and navigate to "http://127.0.0.1:8080/webdirect".



This sample shows how you can combine controls to provide a friendly and modern user interface

Customer Code	1
First Name	John
Last Name	Doe
Street	Murphy Canyon Road
City	San Diego
State	CA
Zip Code	92123
PhoneNun	+1 619 453 0914
CellphoneNum	0

```
...

SCREEN SECTION.

...

05  Push-Button
    col 32 line 4.5
    lines 1 size 8
    css-base-style-name "btn btn-success"
    css-style-name "btn-sm"
    css-icon "fa-save"
    self-act
    exception-value 9
    title "Save"
    .
05  Push-Button
```

Guidelines for writing a web application

WebDirect allows you to bring GUI COBOL programs into the web without major modifications.

Each COBOL program with a Screen Section containing graphical controls can run as a web application with WebDirect.

However, not all GUI features are supported by WebDirect. If you plan to bring an existing COBOL application into the web it is strongly suggested you compile all sources with the *-wd2* option. When using this option, the isCOBOL Compiler will alert you with warning messages if an unsupported feature is being used. For example, the following Frame definition

```
           03 fr frame
              line 2, col 2
              lines 10 size 30
              height-in-cells
              width-in-cells
              full-height
              fill-color 2, fill-percent 50
```

Will produce the following warning at compile time:

```
--W: #179 WD2: Unsupported  FULL-HEIGHT in FRAME control;
--W: #179 WD2: Unsupported  FILL-PERCENT in FRAME control;
```

More details about the unsupported features are provided in Known limitations and differences between Swing and WebDirect.

In order to produce a fast web application, it is strongly suggested you:

- Reduce the number of controls in the screen
- Avoid using embedded and event procedures if not necessary

Known limitations and differences between Swing and WebDirect

This chapter lists the features that are currently not supported by WebDirect as well as behaviors that are different between running as a standard COBOL application and running as a web application.

The list is updated to the date this document has been written.

Please consider that the number of unsupported features decreases as the product becomes more mature.

Most of the unsupported features will just be ignored and the application will behave as if they were not specified in the source code. In some rare cases, an unsupported feature may cause an error.

BAR

In the BAR control the following styles are not supported: DOTTED, DASHED and DOT-DASH.

BITMAP

In the BITMAP control BITMAP-START, BITMAP-END and BITMAP-TIMER properties are not supported and the following mouse events are not returned: MSG-MOUSE-ENTERED, MSG-MOUSE-EXITED, MSG-MOUSE-CLICKED, MSG-MOUSE-DBLCLICK.

CHECK-BOX

In the CHECK-BOX control the following styles are not supported: LEFT-TEXT, VTOP, MULTILINE, FLAT, FRAMED, UNFRAMED, SQUARE.

Check-boxes have a default layout that cannot be altered.

The TITLE-POSITION and BITMAP-DISABLED properties are not supported.

COMBO-BOX

In the COMBO-BOX control the 3-D style is not supported.

MASS-UPDATE has no effect on loading items.

The NOTIFY-DBLCLICK style is not supported and the CMD-DBLCLICK event is not returned.

The list of Combo-Box items is never horizontally truncated. If the item text is too long, then the list width will be greater than the Combo-Box width to ensure that items text is displayed entirely.

The configuration properties *iscobol.gui.curr_bcolor* and *iscobol.gui.curr_fcolor* have no effect.

You can type something when the focus is on a DROP-LIST Combo-Box in order to change the selection. The Combo-Box selects the first item whose value begins with the digit that you typed. In WebDirect only the first letter is evaluated, there's no buffering of digits typed quickly.

The height of the list is controlled by the number of items, not by the Lines property.

When Item-Height is set, it affects both the height of the text-area and the height of elements in the list.

DATE-ENTRY

In the DATE-ENTRY control the following styles are not supported: NO-F4, RIGHT-ALIGN, SHORT-DATE, NO-UPDOWN, SHOW-NONE, SPINNER (that is default for ZK), DECORATION-BACKGROUND-VISIBLE, DECORATION-BORDERS-VISIBLE and WEEK-OF-YEAR-VISIBLE.

The following properties are not supported: CALENDAR-FONT, BITMAP-HANDLE, BITMAP-WIDTH, BITMAP-NUMBER, DECORATION-BACKGROUND, SUNDAY-FOREGROUND, WEEKDAY-FOREGROUND and MAXDAY-CHARACTERS.

ENTRY-FIELD

In the ENTRY-FIELD control the following styles are not supported: AUTO, EMPTY-CHECK, NO-BOX, NO-WRAP, SPINNER, USE-RETURN, USET-TAB and VSCROLL.

The following properties are not supported: ACTION, AUTO-DECIMAL, CURSOR, CURSOR-COL, CURSOR-ROW, MAX-LINES, PROPOSAL-DELAY, SELECTION-TEXT, SELECTION-START, SELECTION-START-ROW, SELECTION-START-COL, FORMAT-STRING and FORMAT-TYPE.

The PLACEHOLDER implementation is a little different than the Swing implementation. In Swing the placeholder text disappears at the first digit from the user, while in WebDirect it disappears as soon as the field gets the focus.

The NTF-CHANGED event is fired only when the user stops typing data into the field. For this reason, there's no point in setting the property NOTIFY-CHANGE-DELAY.

FRAME

In the FRAME control FILL-COLOR2 and FILL-PERCENT properties are not supported as well as the ALTERNATE and FULL-HEIGHT styles.

GRID

In the GRID control the following events are not fired: MSG-BEGIN-ENTRY produced by the Enter key (the user must double click with the mouse in order to produce such event), MSG-BEGIN-ENTRY produced by typing text while the cell is not in edit mode (note that if you wish to provide direct editing, without the need of double clicking on the cell, you can display ENTRY-FIELDs within GRID cells as shown in the installed example), MSG-BEGIN-DRAG, MSG-BEGIN-HEADING-DRAG, MSG-COL-WIDTH-CHANGED, MSG-END-DRAG, MSG-END-HEADING-DRAG, MSG-GOTO-CELL, MSG-GOTO-CELL-DRAG, MSG-GOTO-CELL-MOUSE on the current cell (clicking on the current cell doesn't fire the event; the event is fired when you click on another cell) and MSG-HEADING-DRAGGED.

When the GRID component doesn't have the focus and the user clicks on a cell, the MSG-GOTO-CELL-MOUSE event may not be fired along with the CMD-GOTO event.

The ADJUSTABLE-ROWS and REORDERING-COLUMNS styles are not supported.

The following properties are not supported: ACTION-HIDE-DRAG, CURSOR-FRAME-WIDTH, DRAG-BACKGROUND-COLOR, DRAG-COLOR, DRAG-FOREGROUND-COLOR, END-COLOR, ENTRY-REASON, FINISH-REASON, HEADING-DIVIDER-COLOR, HSCROLL-POS, ROW-DIVIDERS, ROW-HIDING and VSCROLL-POS.

It's not possible to move the cursor from a cell to another using the arrow keys.

The COLUMN-HIDING property is supported only along with the COLUMN-HEADINGS style, you cannot hide columns of a grid without headings. Setting the VIRTUAL-WIDTH property to a value that is less than the Grid size in order to hide the last column (or columns) has no effect in WebDirect.

Modifying CURSOR-X and CURSOR-Y changes the cursor position on the screen only when the Grid gets the focus.

A vertical cursor bar is always visible in the selected cell, even if you're not editing the cell content. This is because a grid cell that is not in edit mode is emulated by a ZK read-only text field and this kind of field always shows the cursor.

Only the first value specified in COLUMN-DIVIDERS and ROW-DIVIDERS is used. Also, the default divider color is light gray.

It's not possible to vertically merge cells in the column heading via the CELL-ROWS-SPAN property if the GRID control has SORTABLE-COLUMNS style or ADJUSTABLE-COLUMNS style.

Scroll-bars are always shown when columns exceed the Grid's size. This is because in WebDirect it is not possible to move among cells using the keyboard, so, without scroll-bars, the columns over the Grid's size would not be reachable.

The HEADING-MENU-POPUP button is shown over each single column. It's possible to show and hide columns, but the entries 'Export...' and 'Copy' are not available.

It's not possible to copy Grid content to the clipboard via the ACTION property.

TILED-HEADINGS may be rendered differently by some browsers.

When you add a record to a grid, the numeric items in the record show leading zeros also if the column data type is '9' with Swing. With WebDirect, instead, the data type '9' removes leading zeros.

ROW-CAPACITY is supported in WebDirect, but the returned value is not always accurate. The ZK Framework doesn't offer a way of knowing how many rows are currently displayed in the grid component. web Direct will estimate that number by dividing the grid height by the row height, and subtracting the number of column headings from the result. Partially visible rows could be left out because of this.

JAVA-BEAN

In WebDirect only the controls of the ZK Framework can be used as JAVA-BEAN; Swing controls are not supported.

In the JAVA-BEAN control the following styles are not supported: HAS-BITMAP, HSCROLL, VSCROLL, NO-BOX, BOXED, 3-D, USE-RETURN and USE-ALT.

As a consequence of the lack of HAS-BITMAP, the BITMAP-HANDLE and BITMAP-WIDTH properties are not supported.

If you want to force the focus on the Java-Bean by invoking the setFocus() method, you have to invoke this method in the Java-Bean BEFORE PROCEDURE.

LABEL

In the LABEL control the VERTICAL style is not supported.

LIST-BOX

In the LIST-BOX control THUMB-POSITION is not supported.

It's not possible to copy List-Box content to the clipboard via the ACTION property.

PUSH-BUTTON

In the PUSH-BUTTON control the following styles are not supported: DEFAULT-BUTTON, FRAMED, MULTILINE, SQUARE and UNFRAMED.

In WebDirect Push-Button titles are always shown entirely. If the Push-Button SIZE is not sufficient to store the title text, then the Push-Button is automatically extended by the Framework.

RADIO-BUTTON

In the RADIO-BUTTON control the following styles are not supported: 3-D, LEFT-TEXT, VTOP, MULTILINE, FLAT, FRAMED, UNFRAMED and SQUARE.

Radio-Buttons have a default layout that cannot be altered.

The TITLE-POSITION and BITMAP-DISABLED properties are not supported.

TAB-CONTROL

In the TAB-CONTROL the following styles are not supported: MULTILINE, BUTTONS, FIXED-WIDTH, BOTTOM, HOT-TRACK, FLAT-BUTTONS and NO-DIVIDERS.

After Procedure and Before Procedure are not supported. The Tab-Control never gets the focus, so it's not possible to change the page by pressing TAB to activate the Tab-Control and then using left and right arrow keys. Use the mouse to change the page in a Tab-Control.

TREE-VIEW

In the TREE-VIEW control the SHOW-SEL-ALWAYS style is not supported. The following properties are not supported: ACTION, BITMAP-NUMBER, BITMAP-HANDLE, BITMAP-WIDTH, ACTION, MASS-UPDATE.

The NEXT-ITEM property doesn't support TVNI-FIRST-VISIBLE, TVNI-NEXT-VISIBLE and TVNI-PREVIOUS-VISIBLE.

No editing is allowed and the following events are not returned: MSG-TV-EXPANDING, MSG-TV-SELCHANGING, MSG-BEGIN-ENTRY, MSG-CANCEL-ENTRY and MSG-FINISH-ENTRY.

The selection color covers the whole line in WebDirect, while in Swing it covers only the item text.

WINDOW

Windows must be Initial/Standard, Independent or Floating.

Initial/Standard windows lose their decoration unless they have the RESIZABLE style; their title becomes the title of the web page also if they're invisible.

Subwindows are not supported.

SHADOW, SCROLL and POP-UP AREA are not supported.

The LM-ZOOM layout manager is not supported.

The CMD-ACTIVATE event is not fired.

The UNDECORATED style just reduces the border, but the title bar is still visible.

Unsupported Controls

The RIBBON, the SCROLL-BAR, the SLIDER, the STATUS-BAR and the WEB-BROWSER controls are not supported in WebDirect.

Other Differences with Desktop Applications

It's good practice to map the internal fonts to existing fonts by setting the following properties in the configuration file:

- iscobol.font.default *
- iscobol.font.fixed *
- iscobol.font.large *
- iscobol.font.medium *
- iscobol.font.small *
- iscobol.font.traditional *

If these fonts are not mapped, WebDirect will use fonts that are surely available, such as Arial, but this will make the GUI of the COBOL application looking different than you expect.

All keyboard input is trapped by the web-browser and therefore keystrokes mapped to exception or termination values will not work. The key that allows the user to move between controls is TAB (use SHIFT+TAB to move to the previous control). Only function keys from F1 to F12 that are not trapped by the web-browser will cause an ACCEPT to terminate with an exception. The other function keys and other keys will not, so it is strongly suggested that you provide a graphical push-button for each mapped keystroke that the COBOL program expects.

An ACCEPT can be terminated by pressing ENTER unless the focus is on a Push-Button, a multiline Entry-Field, a Grid or a Tab-Control.

Relative columns in Screen Section are not supported by WebDirect because the Framework doesn't know the real font size that the browser will use to paint the field. When working on the desktop with Java Swing controls, isCOBOL first puts the control in the window and then inquires its actual size. A similar operation would be very slow if executed by AJAX in the browser, and therefore is better for the user to specify the size and the column of the control using fixed values.

HTML rendering is supported only in the TITLE of Label, Check-Box, Push-Button and Radio-Button as well as in Grid cells when they're not editable.

The CALL CLIENT statement is not supported in WebDirect.

Unsupported Library Routines and functions

In WebDirect all the library routines and functions work on the server machine. Generally speaking all the functions that need to show a dialog on the client machine or to manage files on the client machine are not supported. The following table lists in detail the routines that are partially or totally not supported.

Routine	Notes
A$CURRENT-USER A$GET-USER A$GETTHREAD A$LIST-LOCKS A$LIST-USERS A$USERINFO AS$COPY	Not supported
C$COPY C$DELETE	"@[DISPLAY]:" in the file name is not supported
C$EASYOPEN C$GETCGI C$GUICFG C$OPENSAVEBOX	Not supported
C$SYSTEM	The CSYS-DESKTOP flag is not supported
CBL_READ_SCR_CHARS CBL_READ_SCR_CHATTRS CBL_WRITE_SCR_CHARS CBL_WRITE_SCR_CHATTRS J$GETFROMLAF KEISEN KEISEN1 KEISEN2 KEISEN_SELECT P$CLEARDIALOG P$DISPLAYDIALOG P$ENABLEDIALOG P$SETDIALOG W$CAPTURE W$CENTER_WINDOW	Not supported
W$FONT	The WFONT-CHOOSE-FONT function is not supported
W$HINT W$KEYBUF	Not supported
W$MENU	The WMENU-NEW-TRAY function is not supported
W$MOUSE	All settings of the SET-MOUSE-SHAPE function are supported except a custom mouse pointer
W$PALETTE	The WPALETTE-CHOOSE-COLOR function is not supported
W$PROGRESSDIALOG WIN$PLAYSOUND	Not supported

Routine	Notes
WIN$PRINTER	The WINPRINT-SETUP function is not supported

Note - when a routine that is not supported is called, the effect are unpredictable.

Developing a hello world application from scratch

The next chapter illustrates the steps to create a hello world application from scratch, compile it, deploy it and eventually debug it.

Writing the source

Programs for the web are standard COBOL programs. The following source code produces a screen with a button with "Hello World" inside:

```
       PROGRAM-ID. HELLO.
       SCREEN SECTION.
       01   SCREEN1.
        03  PUSH-BUTTON
           LINE   4
           COL    4
           SIZE 15 CELLS
           TITLE "Hello World"
           EXCEPTION-VALUE 100
         .
       PROCEDURE DIVISION.
       MAIN.
           DISPLAY STANDARD GRAPHICAL WINDOW.
           DISPLAY SCREEN1.
           ACCEPT  SCREEN1 ON EXCEPTION CONTINUE.
```

Compiling the source

Since we plan to debug the program after the deployment, we'll use the -d option.

The -wd2 option is also used to be sure that our program is compatible with WebDirect.

```
iscc -d -wd2 hello.cbl
```

Creating the configuration file

In order to run with WebDirect we must instruct the program to use a specific guifactory class.

In addition, the license codes for the isCOBOL Framework and WebDirect must be provided, so our configuration file will look like this:

```
iscobol.guifactory.class=com.iscobol.gui.client.zk.GuiFactoryImpl
iscobol.license.2021=XXXXXXXXXXXXXXXXXXXXXXX
iscobol.eis.license.2021=XXXXXXXXXXXXXXXXXXXXXXX
```

The configuration file will be placed between program classes in the webapp directories. However, the configuration is also loaded from \etc directory and from the user home directory depending on the drive where Tomcat was started and on the user that owns its process.

Deploying in Tomcat

The easiest way to deploy a new web app is to:

- Deploy the WebDirect sample program as explained in Running the sample application chapter
- Make a copy of the tomcat/webapps/webdirect folder and rename the copy to the name of your choice (i.e. 'test')
  Note - the subfolders "pdf" and "upload" are specific for the WebDirect sample; you can either delete them or rename them, depending on your needs. The other subfolders instead should be left unchanged.
- Add your class files to one of the following:

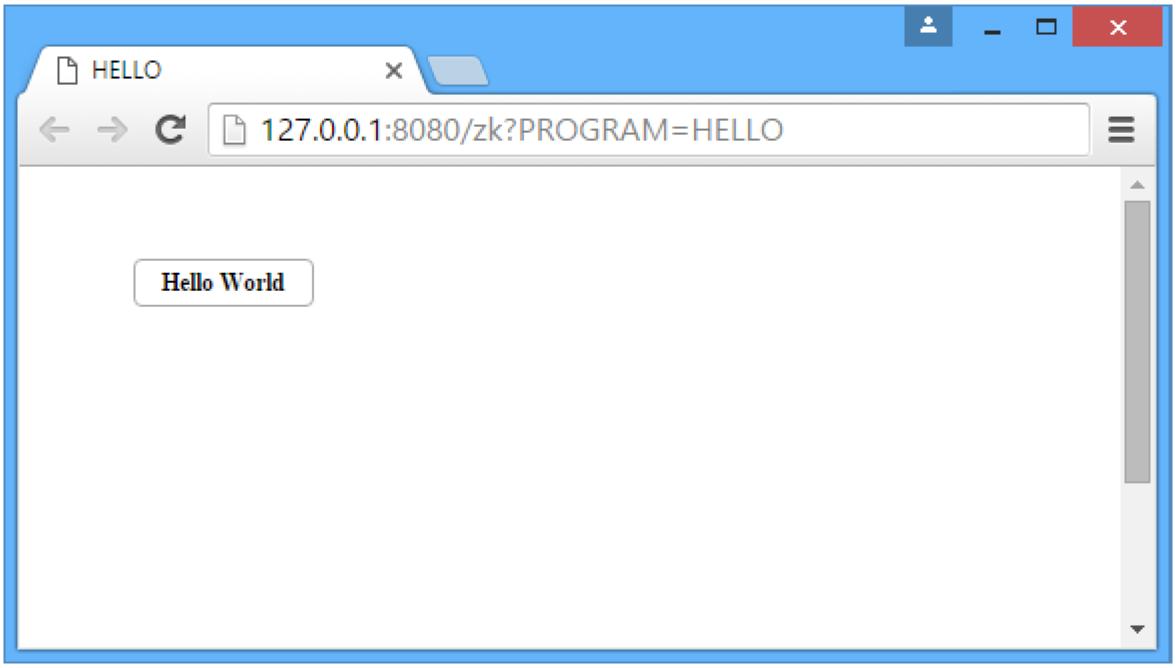
  o one of the folders listed in *iscobol.code_prefix* configuration property
  or, if *iscobol.code_prefix* is not set,

  o the WEB-INF/classes folder

  o a jar file placed in the WEB-INF/lib folder
- Add your properties to the WEB-INF/classes/iscobol.properties file
- Restart Tomcat

Running the application

Run the HELLO program from a web browser using the following URL: http://127.0.0.1:8080/test/zk/IsMainZK?PROGRAM=HELLO to have:



Debugging

In order to debug the web application

- Programs must be compiled with -d option

- The following entry must appear in the configuration

```
iscobol.rundebug=2
```

- The Remote Debugger feature is used

When you connect to the page of your application you will see a blank page. It means that the web application is waiting for the Debugger to connect. Launch the following command to use the Debugger:

```
iscrun -J-Discobol.debug.code_prefix=sourcePath -d -r serverIp
```

Where:

- *sourcePath* is the list of paths where program source code and copyfiles can be found
- *serverIp* is the ip (or name) of the web-server where the web application is running, in our case: 127.0.0.1

If everything has been done correctly, you should see the web page show up while you debug the DISPLAY statements.

Using Native Libraries in isCOBOL EIS WebDirect

Usually DCI and other file handlers provide a file connector solution. When a file connector is available, it's preferable to use it instead of using native libraries. In order to use the file connector you just set the iscobol.file.index and iscobol.file.connector.program(.connector_name) properties to proper values in the iscobol.properties file installed in your webapp.

If a file connector is not available or you have to use other native libraries for features not related to file handling, proceed as follows:

- If the servlet container (Tomcat) is running on Windows, the folder containing the native library must appear in the PATH (System PATH setting, not User PATH). Alternatively, you can copy the necessary native libraries into the Tomcat bin folder.
- If you're working on UNIX/Linux, instead, ensure that the directory containing the native library is listed in the library path (e.g. LD_LIBRARY_PATH, LIBPATH, SHLIB_PATH, etc.)

For example, in a typical configuration /etc/tomcat7/tomcat7.conf sources /usr/share/tomcat7/bin/setenv.sh which is the appropriate place to set global CLASSPATH and LD_LIBRARY_PATH for Tomcat. In some cases, you can also set variables in $HOME/.tomcatrc.

If you're using a container different than Tomcat, consult the documentation for the specific product.

Note that it’s not possible to have more than one WebDirect web application using the same native library. The following error is returned when two web applications try to load the same native library:

```
java.lang.UnsatisfiedLinkError: Native Library library_name already loaded in
another classloader
```

Using c-treeRTG in web applications in Tomcat

Despite it includes native libraries, c-tree is suitable to be used by multiple web applications in the same servlet container without issues. The following steps describe how to set it up in Tomcat.

- copy *iscobol.jar* (that includes the ctreej file handler) to the *WEB-INF/lib* folder of your webapps
- copy *ctree-rtg.jar* (that includes the c-tree client implementation) to the *lib* folder of Tomcat

- have the ctree library available in the library path:
    - If Tomcat is running on Windows, the folder containing the native library must appear in the PATH (System PATH setting, not User PATH). Alternatively, you can copy the necessary native libraries into the Tomcat *bin* folder.
    - If Tomcat is running on UNIX/Linux, instead, ensure that the directory containing the native library is listed in the library path (e.g. LD_LIBRARY_PATH, LIBPATH, SHLIB_PATH, etc.)
- assign your indexed files to the ctreej file handler by setting the iscobol.file.index configuration property to the value “ctreej”.

If you're using a container different than Tomcat, consult the documentation for the specific product.

How to receive parameters in WebDirect

Programs can receive parameters from the URL.

Parameters must be added at the end of the URL using the syntax “&PARAM*N*=Value” (where *N* is a progressive number) and they're intercepted by the COBOL program as chaining parameters.

The following COBOL program, for example, expects 2 parameters, p1 and p2:

```
       PROGRAM-ID. prog.
       WORKING-STORAGE SECTION.
       77 p1 pic x(10).
       77 p2 pic x(10).
       PROCEDURE DIVISION chaining p1 p2.
       main.
           display message "p1=" p1.
           display message "p2=" p2.
           goback.
```

The parameters will be passed through HTTP using GET or POST methods.

Example of GET

The following URL passes two parameters named PARAM1 and PARAM2:

```
http://127.0.0.1:8080/webdirect/zk/IsMainZK?PROGRAM=PROG&PARAM1=AAA&PARAM2=BBB
```

Example of POST

The following HTML form passes two parameters named PARAM1 and PARAM2:

```
<html>
   <form method="POST" action="zk/IsMainZK?PROGRAM=PROG">
    <input type="text" placeholder="Param1" name="PARAM1"/>
    <input type="text" placeholder="Param1" name="PARAM2"/>
    <input type="submit" value="Chiama COBOL"/>
   </form>
</html>
```

Note - the name of the parameter in the URL can be different than the name used by the COBOL program. Parameters are passed according to their ordinal position.

How to Handle Program Exit

By default, when the program terminates due to a GOBACK statement, the last screen remains in the web browser, but is no longer active. This may result in the impression that the program hung, while it was just terminated.

The proper way to handle the program exit is by redirecting the browser to a different web page. This may be the page from which the application was launched, the home page of your website or another of your choosing.

This objective is achieved through JavaScript.

In order to make WebDirect execute JavaScript code:

- define a variable in the Working-Storage Section

```
       77 MY-JAVA-SCRIPT PIC X ANY LENGTH
                         VALUE '<script type="text/javascript">
      -                        'form = document.createElement("form");
      -                        'form.method = "GET";
      -                        'form.action = "https://www.veryant.com";
      -                        'form.target = "_self";
      -                        'document.body.appendChild(form);
      -                        'form.submit();
      -                        '</script>'.
```

  The above code redirects the browser to Veryant's home page. Change the URL according to your needs.

- In the Procedure Division, call WD2$RUN_JS, passing this variable, when you want the JavaScript to be executed:

```
CALL "WD2$RUN_JS" USING MY-JAVA-SCRIPT
```

When a program is running and the user closes the browser window or someone stops the web or application server, an exception with value 91 in crt-status is sent to program in order to terminate the ACCEPT.

Note: always remember to use GOBACK instead of STOP-RUN to make the program exit.

How to Handle Event Lists

EVENT-LIST and EXCLUDE-EVENT-LIST properties work differently in WebDirect environment.

if EXCLUDE-EVENT-LIST = 1:

- if EVENT-LIST is empty ALL EVENTS are NOT SENT to the program.
- if EVENT-LIST is not empty:
- the events in the EVENT-LIST are NOT SENT to the program.
- the events NOT in the EVENT-LIST are SENT to the program.

if EXCLUDE-EVENT-LIST = 0:

- if EVENT-LIST is empty ALL EVENTS are SENT to the program.
- if EVENT-LIST is not empty:
- the events in the EVENT-LIST are SENT to the program.
- the events NOT in the EVENT-LIST are NOT SENT to the program.

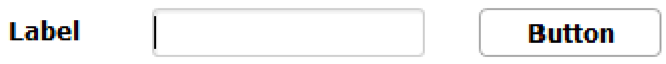
Customize the WebDirect Layout using CSS

Like all web sites and web applications, the layout of programs running with WebDirect can be customized using CSS (Cascading Style Sheets).

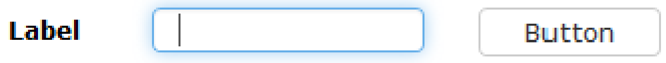
ZK8 default and alternative themes

By default the COBOL screens are rendered using the ZK8 default theme.

The following screenshot shows how a simple screen including a label, a text field and a push-button appears in a web browser with the default ZK8 theme.



It is possible to change this theme by setting the iscobol.wd2.style configuration property. The following screenshot shows how the same screen as above is shown when *iscobol.wd2.style=bs* (Bootstrap styling)



Custom Style Association

The file iscobol.css provided along with isCOBOL should always be stored in the resources/css folder of any WebDirect application. This file contains the default styling for the COBOL graphical controls. Editing or omitting this file may produce a bad layout for the web application.

The default styling can be customized by writing a new css file, putting it in the resources/css folder of the web application and point to it by setting the configuration property iscobol.wd2.additional_stylesheet.

The css file must have the following syntax:

```
<css-style-name> {
<attribute>:<value>;
...
<attribute>:<value>;
}
```

In order to associate a particular style to a graphical control, you take advantage of the CSS-BASE-STYLE-NAME and CSS-STYLE-NAME properties, that are supported for all controls.

These properties take a string parameter that specifies the style name.

Different controls can use the same style.

CSS allow you to create effects that would not be possible using the COBOL language. In the example below, all GUI controls with CSS-STYLE-NAME="highlite" will be highlighted with a shadow when the mouse pointer goes over them.

Content of the custom css file:

```
.highlite:hover{
 box-shadow: 10px 10px 5px #888888;
 transition: all 0.2s;
}
```

content of COBOL program Screen Section:

```
    01 SCR-SAMPLE.
       05 CHECK-BOX ... CSS-STYLE-NAME "mystyle" ....
       05 RADIO-BUTTON ... CSS-STYLE-NAME "mystyle" ....
```

CSS-BASE-STYLE-NAME acts similarly to CSS-STYLE-NAME, in which it allows one or more css classes to be applied to the control, but differs from it because specifying a css class in the CSS-BASE-STYLE-NAME will overwrite any isCOBOL default classes, allowing a completely customized style to be applied. On the other hand, using CSS-STYLE-NAME to supply a css class to the control, will cause the runtime to append the specified class name to the default one supplied by the environment. If you wish to completely overwrite default styling, use CSS-BASE-STYLE-NAME, and if you want to complement the default styles, use CSS-STYLE-NAME.

Both properties can be used simultaneously. For example, to completely customize the look of your application, use CSS-BASE-STYLE-NAME to overwrite the COBOL styles and provide your own styling to each control class (i.e. buttons, entry-fields, etc.), and CSS-STYLE-NAME to provide specific styling for a specific control's purpose.

Example

```
   01 SCR-SAMPLE.
      05 PUSH-BUTTON ... CSS-BASE-STYLE-NAME "my-btn" CSS-STYLE-NAME "ok-btn" ....
      05 PUSH-BUTTON ... CSS-BASE-STYLE-NAME "my-btn" CSS-STYLE-NAME "cancel-btn" ....
```

Fonts and colors set by the COBOL program have priority over fonts and colors set by the style associated to the CSS-STYLE-NAME property, unless you put the *!important* clause after the css entries. Fonts and colors set by the COBOL program are overridden by fonts and colors set by the style associated to the CSS-BASE-STYLE-NAME property instead.

When iscobol.wd2.style is set to "bs", you have available Bootstrap css classes for easy styling of the application. For example, to create OK and CANCEL buttons, you could use the following:

```
        03 PUSH-BUTTON
           LINE 22, COL 2 LINES 2 SIZE 10 CELLS
           CSS-BASE-STYLE-NAME "btn"
           CSS-STYLE-NAME "btn-success"
           OK-BUTTON
           .
        03 PUSH-BUTTON
           LINE 22, COL 13 LINES 2 SIZE 20 CELLS
           CSS-BASE-STYLE-NAME "btn"
           CSS-STYLE-NAME "btn-danger"
           CANCEL-BUTTON
           .
```

Where *btn-success* and *btn-danger* are two of the available CSS classes provided by Bootstrap. See http://getbootstrap.com/components for the list of available classes.

With the above code the OK button will have a green color and the Cancel button will have a red color.

You can use your web-browser development features to check styles and classes that were applied to the Screen Section elements.

Font Awesome icons on Push-Buttons

For the Push-Button controls, the CSS-ICON is also available. This alphanumeric property allows you to access Font Awesome icons when the application is deployed as WebDirect. The icon list is available at the following web page: https://fontawesome.com/icons?from=io.

Note - ZK integrates Font Awesome 4.0.1, therefore some of the icons listed in the above page might not be available.

For example, to have a Push-Button with a checkmark, all is needed is to find the relevant icon from the list at the above web page (in this case is https://fontawesome.com/icons/check?style=solid&from=io), copy the icon class name, in this case "fa-check", and paste it in the CSS-ICON property of the PUSH-BUTTON, e.g.

```
       05 PUSH-BUTTON ... CSS-ICON "fa-check"...
```

At runtime the chosen icon will be displayed.

Note - CSS-ICON replaces the icon set by BITMAP-HANDLE property, if any.

Chapter 7

Troubleshooting

This chapter lists the most common errors that may appear while working with isCOBOL EIS.

Tomcat startup errors

If a connection error occurs and the browser cannot load the page with the COBOL application, ensure that Tomcat is correctly started.

Information on Tomcat startup errors can be found in catalina.currentdate.log file in Tomcat's logs directory.

Blank page with EIS WD2

If an empty blank screen appears in place of the COBOL application, it could mean that WD2 could not initialize the program correctly. Error messages that help troubleshooting the cause of the problem can be found in the stdout_currentdate.log, stderr_currentdate.log and localhost.currentdate.log files in Tomcat's logs directory.

"Missing License" is a common problem that causes blank screen. Check that the *iscobol.eis.license.2019* property is set in /etc/iscobol.properties or in the web application's WEB-INF/classes/iscobol.properties file.

The blank page may also be caused by the application waiting for Debugger, if iscobol.rundebug property is set in the configuration.

Also, the blank page may be caused by the web application terminating before the first DISPLAY, for example due to i/o errors. Remote debugging can help in this case.

HTTP errors

When an error occurs in the web application, it usually causes HTTP ERRORS like 404 and 500.

In order to retrieve the full Exception stack, consult the log files in Tomcat's *logs* directory.

Preventing errors related to the UI in EIS Servelt and Web Service environments

It can happen that a COBOL program performs a DISPLAY or ACCEPT on the user interface in a EIS environment. This could happen, for example, if you reuse existing code that was previously used in a Desktop environment. These operations are not supported by Servlets and Web Services and may lead to unexpected behaviors and errors. In order to prevent these error conditions, Veryant recommends to add -whttp to your compiler options when you compile programs for the EIS environment.

Chapter 8

Tomcat Installation

Introduction

In order to host isCOBOL EIS COBOL Servlets, you need to install and run a Servlet container. There are many Servlet containers available.

You can see lists and comparisons of Servlet containers at http://en.wikipedia.org/wiki/Comparison_of_web_servers. In the *Features* table, search for "Yes" in the *Java Servlets* column.

Veryant has tested and recommends Apache Tomcat 8.5 or above.

Download and install Tomcat

The Apache Tomcat main page is http://tomcat.apache.org/ .

Here are some steps to download and install Tomcat 8.5 on Windows:

NOTE - To avoid problems, uninstall earlier versions of the Tomcat service before installing Tomcat 8.5

- Make sure that you already have installed JDK 8 (or greater) and isCOBOL Evolve
- Visit https://tomcat.apache.org/download-80.cgi.
- Find the Binary Distributions section and click on the *32-bit/64-bit Windows Service Installer* link
- Run the downloaded executable file and follow the prompts accepting the defaults

Configure Tomcat to use the isCOBOL EIS framework

$CATALINA_HOME is the Tomcat installation directory. The default location on Windows is:

```
C:\Program Files\Apache Software Foundation\Tomcat 8.5
```

To configure Tomcat to use the isCOBOL Runtime Framework on Windows you can change the value of the shared.loader property in *$CATALINA_HOME/conf/catalina.properties* to the following:

```
shared.loader=/program\ files/veryant/iscobol2021R1/lib/iscobol.jar
```

On Unix, set the CLASSPATH in Tomcat's startup environment to include iscobol.jar. For example, on Linux add the following line to /etc/tomcat8/tomcat8.conf or other script called during the Tomcat startup:

```
CLASSPATH=$ISCOBOL/lib/iscobol.jar:$CLASSPATH; export CLASSPATH
```

Make sure that you have a valid license for isCOBOL Evolve in /etc/iscobol.properties (i.e. iscobol.license.<release year>=<license key>) or in the iscobol.properties in the home directory for the user that starts Tomcat.

Disable persistence across restarts

Whenever Apache Tomcat is shut down normally and restarted, or when an application reload is triggered, the standard Manager implementation will attempt to serialize all currently active sessions to a disk file located via the pathname attribute. All such saved sessions will then be deserialized and activated (assuming they have not expired in the mean time) when the application reload is completed. In order to successfully restore the state of session attributes, all such attributes must implement the java.io.Serializable interface. Since not all isCOBOL classes are serializable, we strongly suggest you disable this persistence feature.

To disable this persistence feature, create a Context configuration file for your web application and add the following element there:

```
<Manager pathname="" />
```

Note - The file *context.xml* in the Tomcat home directory already includes the above entry, but it's commented. You can easily disable the persistence for all your web applications by removing the comment markers around the entry.

Data access

When relying on file handler that includes native parts (e.g. DCI, c-tree or Vision) we suggest using a File Connector, if available.

However, the c-tree file handler is also certified to work without the need of a File Connector if installed and configured as follows:

- copy *iscobol.jar* to the webapp's WEB-INF/lib folder
- copy *ctree-rtg.jar* to the Tomcat's lib folder
- set *iscobol.file.index=ctreej* in the configuration

Relative paths

Relative paths used by COBOL programs in EIS environment are relative to the webapp directory except for *iscobol.code_prefix* paths that are relative to the Tomcat working directory.

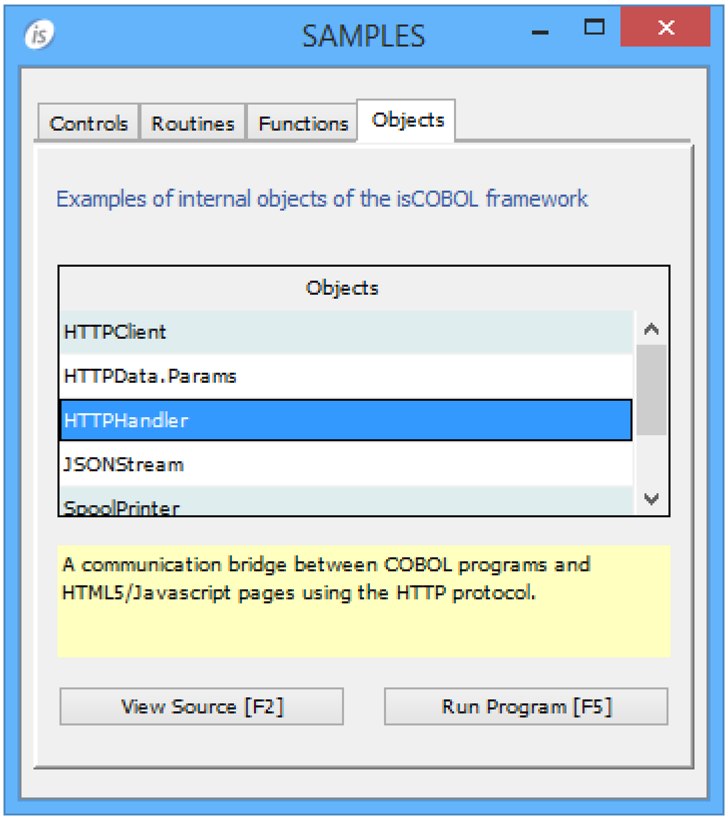
Chapter 9

Appendices

HTTPHandler class (com.iscobol.rts.HTTPHandler)

The HTTPHandler is an internal class that provides a communication bridge between COBOL programs and HTML5/Javascript pages using the HTTP protocol.

A sample program can be found in isCOBOL Samples.



Examples of internal objects of the isCOBOL framework

Objects
HTTPClient
HTTPData.Params
HTTPHandler
JSONStream
SpoolPrinter

A communication bridge between COBOL programs and HTML5/Javascript pages using the HTTP protocol.

Constructor

Creates a new instance of the HTTPHandler class.

General format

```
HTTPHandler
```

General rules

1. A reference to HTTPHandler should be defined in the program's Linkage Section.

Code example

```
...
configuration section.
repository.
    class HTTPHandler  as "com.iscobol.rts.HTTPHandler"
....
linkage section.
77 objHTTPHandler object reference HTTPHandler.
...
procedure division using objHTTPHandler.
...
```

accept

Receives parameters from the HTTP.

General format

```
void accept( params )
```

Syntax rules

1. *params* is a data item for which the IS IDENTIFIED clause has been used.

General rules

1. *params* elements name matches the name of the parameter passed by the HTTP client.
2. If the parameter passed by the HTTP client is a file, then
   - o The file is stored in the folder identified by the iscobol.http.upload.directory * configuration property, whose default is the server temp directory.
   - o The name of the file is returned to the COBOL program, not the file content. The name is formed by five parts:
     - i. the folder where the file has been stored
     - ii. the prefix specified by the configuration property iscobol.http.upload.prefix *, if set
     - iii. a unique prefix automatically generated by the framework in order to avoid duplicate names
     - iv. the underscore character
     - v. the name of the file passed by the client

Example

Consider the following HTML form:

```
<form action="servlet/isCobol(PROG1)" enctype="multipart/form-data" method="post">
Type some text:<br>
<input type="text" name="textline" size="30"><br>
Choose a file to upload:<br>
<input type="file" name="datafile" size="40"><br>
Send to server<br>
<input type="submit" value="Send">
</form>
```

The user types data into the text area and browses for a file on disk.

When the ‘Send’ button is clicked, the COBOL program ‘PROG1’ is called. In order to intercept the field's content, PROG1 should be written as follows:

```
       CONFIGURATION SECTION.
       REPOSITORY.
           CLASS WEB-AREA AS "com.iscobol.rts.HTTPHandler"
           .

       ...
       WORKING-STORAGE SECTION.
       01 http-data identified by "http_data".
          03  identified by "textline".
              05 w-textline pic x any length.
          03  identified by "datafile".
              05 w-datafile pic x any length.
       ...
       LINKAGE SECTION.
       01 LNK-AREA OBJECT REFERENCE WEB-AREA.


       PROCEDURE DIVISION USING LNK-AREA.
       MAIN.
           LNK-AREA:>accept(http-data).
```

After the accept() invocation, *w-textline* is set to the text typed by the user in the text area of the HTML form, while *w-datafile* is set to the name of the uploaded file in the format of <*server_folder*>/ <*random_id*>_<*filename*>.

acceptAllParameters

Receives a list of all parameters followed by their value. This is useful to monitor what is actually passed by the HTTP client.

General format

```
void acceptAllParameters( params )
```

Syntax rules

1. *params* is an alphanumeric data item. It’s good practice to use items with picture X ANY LENGTH for this purpose.

General rules

1. A single buffer is returned by this method. The buffer contains all the parameter names followed by their respective value.

acceptEx

Receives parameters from the HTTP by invoking acceptFromJSON or acceptFromXML depending on the Content-type field. If no Content-type is available in the request header, then it invokes acceptFromJSON or acceptFromXML depending on the iscobol.rest.default_stream configuration setting.

General format

```
void acceptEx( params )
```

Syntax rules

1. *params* is a level 01 data item for which the IS IDENTIFIED clause has been used.

General rules

1. *params* elements name matches the name of the parameter passed by the HTTP client.

acceptFromJSON

Receives parameters from the HTTP assuming that they're passed as a JSON stream.

General format

```
void acceptFromJSON( params )
```

Syntax rules

1. *params* is a level 01 data item for which the IS IDENTIFIED clause has been used.

General rules

1. *params* elements name matches the name of the parameter passed by the HTTP client.
2. The following escapes are allowed: \b, \f, \n, \r, \t, \" and \\. Any other character preceeded by a backslash is considered invalid and makes the read fail unless you set iscobol.jsonstream.allow_backslash_escaping_any_character (boolean) to true in the configuration.

acceptFromXML

Receives parameters from the HTTP assuming that they're passed as an XML stream.

General format

```
void acceptFromXML( params )
```

Syntax rules

1. *params* is a level 01 data item for which the IS IDENTIFIED clause has been used.

General rules

1. *params* elements name matches the name of the parameter passed by the HTTP client.

addOutHeader

Adds an item to the response HTTP header.

General format

```
void addOutHeader( name, value )
```

Syntax rules

1. *name* and *value* are alphanumeric data items or literals.

displayBinaryFile

Returns the content of a binary file as a response to the HTTP client. The file is treated as a sequence of bytes, no unicode conversion is applied.

General format

```
void displayBinaryFile( fileName, mimeType )
```

Syntax rules

1. *fileName* and *mimeType* are alphanumeric data items.

General rules

1. It's good practice to provide a valid MIME type along with the file name.

displayError

Returns a numeric error code to the HTTP client.

General format

```
void displayError( errNum, errText )
```

Syntax rules

1. *errNum is a numeric data item or literal*
2. *errText* is an alphanumeric data item or literal.

General rules

1. You should provide a valid HTTP status code as described in the latest HTTP/1.1 RFC at page 39.

displayHTML

Returns an HTML stream to the HTTP client.

General format

```
void displayHTML( html, docType )
```

Syntax rules

1. *html* is a level 01 data item for which the *IS IDENTIFIED clause* has been used.

2. *docType* is an alphanumeric data item or literal.

General rules

1. *html* data item must be identified by html tags, in particular the item at level 01 must be IDENTIFIED BY "HTML".
2. *docType* specifies the `<!DOCTYPE>` declaration as described here. It might be null.
3. The MIME type "text/html" is automatically applied.

Example

The following COBOL program produces an HTML output with different text styles:

```
       CONFIGURATION SECTION.
       REPOSITORY.
          CLASS WEB-AREA AS "com.iscobol.rts.HTTPHandler"
          .
       WORKING-STORAGE SECTION.
       01  html identified by "html".
           03 identified by "p".
              05 identified by "b".
                 07 bold-text pic x any length.
           03 identified by "p".
              05 identified by "i".
                 07 italic-text pic x any length.
           03 identified by "p".
              05 plain-text pic x any length.

       LINKAGE SECTION.
       01 LNK-AREA OBJECT REFERENCE WEB-AREA.

       PROCEDURE DIVISION USING LNK-AREA.
       MAIN.
           move "bold" to bold-text.
           move "italic" to italic-text.
           move "plain" to plain-text.
           lnk-area:>displayHTML(html, null).
           goback.
```

displayText

Returns raw text to the HTTP client.

General format

```
void displayText( text )
```

Syntax rules

1. *text* is an alphanumeric data item or literal.

General rules

1. The MIME type "text/plain" is automatically applied.

displayTextFile

Returns the content of a binary file as a response to the HTTP client. The file is processed using the current encoding.

General format

```
void displayTextFile( fileName, mimeType )
```

Syntax rules

1. *fileName* and *mimeType* are alphanumeric data items.

General rules

1. It's good practice to provide a valid MIME type along with the file name.

displayEx

Returns a stream to the HTTP client by invoking displayJSON or displayXML depending on the Content-type field. If no Content-type is available in the response header, then it invokes displayJSON or displayXML depending on the iscobol.rest.default_stream configuration setting.

Format 1

```
void displayEx( stream )
```

Format 2

```
void displayEx( stream, hasDummyRoot)
```

Syntax rules

1. *stream* is a level 01 data item for which the *IS IDENTIFIED clause* has been used.
2. *hasDummyRoot* is an alphanumeric data item or literal hosting a boolean value (e.g. "0", "1", "true", "false", "yes", "no", "on" or "off"). If the boolean value is TRUE, then the top level item of Record-Definition is discarded and will not appear in the JSON stream . This parameter can be NULL, in such case, FALSE is assumed.

General rules

1. The MIME type is automatically applied.
2. *hasDummyRoot* is considered only for the JSON type.

displayXML

Returns a XML stream to the HTTP client.

General format

```
void displayXML( xml )
```

Syntax rules

1. *xml* is a level 01 data item for which the *IS IDENTIFIED clause* has been used.

General rules

1. The MIME type “text/xml” is automatically applied.

displayJSON

Returns a JSON stream to the HTTP client.

Format 1

```
void displayJSON( json )
```

Format 2

```
void displayJSON( json, hasDummyRoot )
```

Syntax rules

1. *json* is a level 01 data item for which the *IS IDENTIFIED clause* has been used.
2. *hasDummyRoot* is an alphanumeric data item or literal hosting a boolean value (e.g. "0", "1", "true", "false", "yes", "no", "on" or "off"). If the boolean value is TRUE, then the top level item of Record-Definition is discarded and will not appear in the JSON stream . This parameter can be NULL, in such case, FALSE is assumed.

getContentType

Return the HTTP Content Type of the request

General format

```
String getContentType( )
```

getHeader

Reads the value of a specific item in the HTTP header.

General format

```
String getHeader( name )
```

Syntax rules

1. *name* is an alphanumeric data item or literal.

General rules

1. The header name should be specified in lower case otherwise it’s not found. For example, in order to get the value of "Content-Type", look for "content-type".

getIntHeader

Reads the value of a specific item in the HTTP header assuming that it’s an integer number.

General format

```
int getIntHeader( name )
```

Syntax rules

1. *name* is an alphanumeric data item or literal.

General rules

1. The header name should be specified in lower case otherwise it’s not found. For example, in order to get the value of "Content-Length", look for "content-length".

getMethod

Returns the HTTP request method used for client and server calls.

General format

```
String getMethod( )
```

General rules

1. The returned value is one of the following strings: CONNECT, DELETE, GET, HEAD, OPTIONS, POST, PUT.

getRequest

Return the HTTP Request object instance.

General format

```
Object getRequest( )
```

General rules

1. The returned object should be cast to *javax.servlet.http.HttpServletRequest*.

Example

Retrieve the IP address of the client machine that made the request:

```
repository.
    class HTTPHandler as "com.iscobol.rts.HTTPHandler"
    class HTTPRequest as "javax.servlet.http.HttpServletRequest"
    .
working-storage section.
77 servlet-request object reference HTTPRequest.
77 client-ip pic x any length.
linkage section.
77 http-handler object reference HTTPHandler.
procedure division using http-handler.
    set servlet-request to http-handler:>getRequest()
                        as HTTPRequest.
    set client-ip to servlet-request:>getRemoteAddr().
```

getResponse

Return the HTTP Response object instance.

General format

```
Object getResponse( )
```

General rules

1. The returned object should be cast to *javax.servlet.http.HttpServletResponse*.

Example

Retrieve the Response object instance:

```
repository.
    class HTTPHandler as "com.iscobol.rts.HTTPHandler"
    class HTTPResponse as "javax.servlet.http.HttpServletResponse"
    .
working-storage section.
77 servlet-response object reference HTTPResponse.
linkage section.
77 http-handler object reference HTTPHandler.
procedure division using http-handler.
    set servlet-response to http-handler:>getResponse()
                        as HTTPResponse.
```

getSession

Return the HTTP Session object instance.

General format

```
Object getSession( )
```

General rules

1. The returned object should be cast to *javax.servlet.http.HttpSession*.

Example

Retrieve the unique ID of the current HTTP session:

```
repository.
    class HTTPHandler as "com.iscobol.rts.HTTPHandler"
    class HTTPSession as "javax.servlet.http.HttpSession"
    .
working-storage section.
77 servlet-session object reference HTTPSession.
77 unique-id pic x any length.
linkage section.
77 http-handler object reference HTTPHandler.
procedure division using http-handler.
    set servlet-session to http-handler:>getSession()
                        as HTTPSession.
    set unique-id to servlet-session:>getId().
```

invalidateSession

Invalidates the current HTTP session and removes all session data. This is the correct way to terminate the whole application. This method should be associated with the “Exit” function of your application.

General format

```
void invalidateSession()
```

isRedirect

Tells if a redirect has been issued or not.

General format

```
boolean isRedirect()
```

Code example

```
if objHTTPHandler:>isRedirect()
   |a redirect has been issued
else
   |a redirect has not been issued
end-if
```

isSessionInvalidated

Tells if the current session has been invalidated or not.

General format

```
boolean isSessionInvalidated()
```

Code example

```
if objHTTPHandler:>isSessionInvalidated()
   |the session has been invalidated
else
   |the session is still valid
end-if
```

processHtmlFile

Process an HTML file, replacing items delimited by %% characters with the corresponding COBOL data item.

General format

```
boolean processHtmlFile (htmlFile)
```

Syntax rules

1. *htmlFile* is a level 01 data item for which the IS IDENTIFIED clause has been used.

General rules

1. *htmlFile's* 01 level is identified by the name of the HTML file to be processed while the subsequent variable identifiers are searched for in the file inside the delimiters '%%' (or prefixed by colon, depending on the iscobol.http.value_prefix_colon (boolean) setting). The file is loaded in the directory (or directories) specified by the configuration property iscobol.http.html_template_prefix *.
2. If the file name doesn't end with the extension ".html" nor ".htm" the method appends ".html" to the name. If the file with this name doesn't exist then it appends the extension ".htm". If still the file doesn't exist then the method looks for a file with the exact name.
3. The method returns true if the operation is successful and false otherwise: in the latter case an error message will be included in the HTML output.

Example

Consider having a file named "mypage.html" with the following content:

```
<html>
  <p>This page has been shown by %%cobolname%%</p>
</html>
```

The following COBOL program will display the above HTML with "isCOBOL" instead of "cobolname".

```
       CONFIGURATION SECTION.
       REPOSITORY.
          CLASS WEB-AREA AS "com.iscobol.rts.HTTPHandler"
          .
       WORKING-STORAGE SECTION.
       01  html identified by "mypage".
           03 identified by "cobolname".
              05 cob-name pic x any length.

       LINKAGE SECTION.
       01 LNK-AREA OBJECT REFERENCE WEB-AREA.

       PROCEDURE DIVISION USING LNK-AREA.
       MAIN.
           move "isCOBOL" to cob-name.
           LNK-AREA:>processHtmlFile(html).
```

processHtmlString

Process the HTML code contained in an alphanumeric variable, replacing items delimited by %% characters with the corresponding COBOL data item.

General format

```
boolean processHtmlString (string, params)
```

Syntax rules

1. *string* is an alphanumeric data item or literal.
2. *params* is a level 01 data item for which the IS IDENTIFIED clause has been used.

General rules

1. *params* child variable identifiers are searched for in the *string* text inside the delimiters '%%' (or prefixed by colon, depending on the iscobol.http.value_prefix_colon (boolean) setting).
2. The method returns true if the operation is successful and false otherwise: in the latter case an error message will be included in the HTML output.

Example

The following COBOL program will display an HTML output whose text is “This output has been generated by isCOBOL”:.

```
       CONFIGURATION SECTION.
       REPOSITORY.
          CLASS WEB-AREA AS "com.iscobol.rts.HTTPHandler"
          .
       WORKING-STORAGE SECTION.
       01  params identified by "_".
           03 identified by "cobolname".
              05 cob-name pic x any length.

       LINKAGE SECTION.
       01 LNK-AREA OBJECT REFERENCE WEB-AREA.

       PROCEDURE DIVISION USING LNK-AREA.
       MAIN.
           move "isCOBOL" to cob-name.
           LNK-AREA:>processHtmlString("This output has been generated by
%%cobolname%%", params).
```

redirect

Issues a redirect.

General format

```
void redirect( newPage )
```

Syntax rules

1. *newPage* is an alphanumeric data item or literal.

General rules

1. if *newPage* starts with a protocol (e.g. "http:"), then it is used as is, otherwise it is considered a relative URL and it is appended to the webapp base URL.

setMethod

Sets the HTTP request method used for client and server calls.

General format

```
void setMethod( method )
```

Syntax rules

1. *method* is an alphanumeric data item or literal.

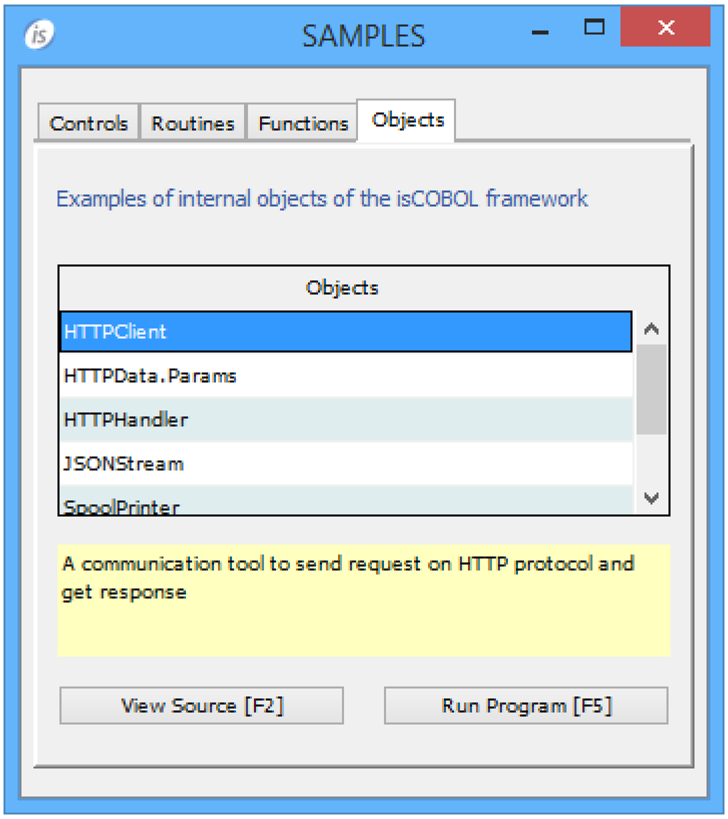
General rules

1. *method* must be one of the following strings: CONNECT, DELETE, GET, HEAD, OPTIONS, POST, PUT.

HTTPClient class (com.iscobol.rts.HTTPClient)

The HTTPClient is an internal class that provides many useful features to communicate with existing HTTP services like Web Service (REST/SOAP) HTTP server etc.

A sample program can be found in isCOBOL Samples.



Examples of internal objects of the isCOBOL framework

Objects
HTTPClient
HTTPData.Params
HTTPHandler
JSONStream
SpoolPrinter

A communication tool to send request on HTTP protocol and get response

Constructor

General Format

```
HTTPClient
```

General rules

A reference to HTTPHandler should be defined in the program's Working-Storage Section.

Code example.

```
...
   configuration section.
   repository.
       class http-client as "com.iscobol.rts.HTTPClient"
...
   working-storage section.
   77 http object reference http-client.
...
   procedure division.
...
   set http to http-client:>new().
...
```

doDelete

Executes an HTTP request using DELETE method.

Format 1

```
void doDelete( strUrl )
```

Format 2

```
void doDelete( strUrl, params )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *params* should contain an *HTTPData.Params* object where HTTP parameters are defined. See HTTPData.Params class (com.iscobol.rts.HTTPData.Params) for more information.

General rules

1. In Format 1, if you need to pass HTTP parameters, you can use the *setParameters*() method.

doDeleteEx

Executes an HTTP request using DELETE method specifying the data stream and having the MIME type set automatically.

Format 1

```
void doDeleteEx( strUrl, content )
```

Format 2

```
void doDeleteEx( strUrl, type, content )
```

Format 3

```
void doDeleteEx( strUrl, type, content, hasDummyRoot )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *content* contains the data stream.
3. *type* is the MIME type (i.e. "text/xml")
4. *hasDummyRoot* is an alphanumeric data item or literal hosting a boolean value (e.g. "0", "1", "true", "false", "yes", "no", "on" or "off"). If the boolean value is TRUE, then the top level item of Record-Definition is discarded and will not appear in the JSON stream . This parameter can be NULL, in such case, FALSE is assumed.

General rules

1. In Format 1, *type* is set to "application/json" or "application/xml" depending on the Content-type request header field. If no Content-type field is available in the request header, then the type is controlled by the iscobol.rest.default_stream configuration setting.
2. If *content* is an appropriately structured variable with external names and the type is set to "application/json", the method makes a request using the JSON format.
3. If *content* is an appropriately structured variable with external names but type is not set to "application/json", the method makes a request using the XML format.

doGet

Executes an HTTP request using GET method.

Format 1

```
void doGet( strUrl )
```

Format 2

```
void doGet( strUrl, params )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *params* should contain an *HTTPData.Params* object where HTTP parameters are defined. See HTTPData.Params class (com.iscobol.rts.HTTPData.Params) for more information.

General rules

In Format 1, if you need to pass HTTP parameters, you can use the *setParameters*() method.

doPatch

Executes an HTTP request using PATCH method.

Format 1

```
void doPatch( strUrl )
```

Format 2

```
void doPatch( strUrl, params )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *params* should contain an *HTTPData.Params* object where HTTP parameters are defined. See HTTPData.Params class (com.iscobol.rts.HTTPData.Params) for more information.

General rules

1. In Format 1, if you need to pass HTTP parameters, you can use the *setParameters*() method.

doPatchEx

Executes an HTTP request using PATCH method specifying the data stream and having the MIME type set automatically.

Format 1

```
void doPatchEx( strUrl, content )
```

Format 2

```
void doPatchEx( strUrl, type, content )
```

Format 3

```
void doPatchEx( strUrl, type, content, hasDummyRoot )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *content* contains the data stream.
3. *type* is the MIME type (i.e. "text/xml")
4. *hasDummyRoot* is an alphanumeric data item or literal hosting a boolean value (e.g. "0", "1", "true", "false", "yes", "no", "on" or "off"). If the boolean value is TRUE, then the top level item of Record-Definition is discarded and will not appear in the JSON stream . This parameter can be NULL, in such case, FALSE is assumed.

General rules

1. In Format 1, *type* is set to "application/json" or "application/xml" depending on the Content-type request header field. If no Content-type field is available in the request header, then the type is controlled by the iscobol.rest.default_stream configuration setting.
2. If *content* is an appropriately structured variable with external names and the type is set to "application/json", the method makes a request using the JSON format.

If *content* is an appropriately structured variable with external names but type is not set to "application/json", the method makes a request using the XML format.

doPost

Executes an HTTP request using the POST method.

Format 1

```
void doPost( strUrl )
```

Format 2

```
void doPost( strUrl, params )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *params* should contain an *HTTPData.Params* object where HTTP parameters are defined. See HTTPData.Params class (com.iscobol.rts.HTTPData.Params) for more information.

General rules

1. In Format 1, if you need to pass HTTP parameters, you can use the setParameter method.

doPostEx

Executes an HTTP request using the POST method specifying the data stream and having the MIME type set automatically.

Format 1

```
void doPostEx( strUrl, content )
```

Format 2

```
void doPostEx( strUrl, type, content )
```

Format 3

```
void doPostEx( strUrl, type, content, hasDummyRoot )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *content* contains the data stream.
3. *type* is the MIME type (i.e. "text/xml")
4. *hasDummyRoot* is an alphanumeric data item or literal hosting a boolean value (e.g. "0", "1", "true", "false", "yes", "no", "on" or "off"). If the boolean value is TRUE, then the top level item of Record-Definition is discarded and will not appear in the JSON stream . This parameter can be NULL, in such case, FALSE is assumed.

General rules

1. In Format 1, *type* is set to "application/json" or "application/xml" depending on the Content-type request header field. If no Content-type field is available in the request header, then the type is controlled by the iscobol.rest.default_stream configuration setting.
2. If *content* is an appropriately structured variable with external names and the type is set to "application/json", the method makes a request using the JSON format.
3. If *content* is an appropriately structured variable with external names but type is not set to "application/json", the method makes a request using the XML format.

doPostMultipart

Executes an HTTP request using the POST method sending the parameters using the multipart/form-data protocol.

General format

```
void doPostMultipart( strUrl, parameters )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *parameters* is an instance of HTTPData.Params class (com.iscobol.rts.HTTPData.Params).

doPut

Executes an HTTP request using the PUT method.

Format 1

```
void doPut( strUrl )
```

Format 2

```
void doPut( strUrl, params )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *params* should contain an *HTTPData.Params* object where HTTP parameters are defined. See HTTPData.Params class (com.iscobol.rts.HTTPData.Params) for more information.

General rules

In Format 1, if you need to pass HTTP parameters, you can use the setParameter method.

doPutEx

Executes an HTTP request using the PUT method specifying the data stream and having the MIME type set automatically.

Format 1

```
void doPutEx( strUrl, content )
```

Format 2

```
void doPutEx( strUrl, type, content )
```

Format 3

```
void doPutEx( strUrl, type, content, hasDummyRoot )
```

Syntax rules

1. *strUrl* is the URL to invoke.
2. *content* contains the data stream.
3. *type* is the MIME type (i.e. "text/xml")
4. *hasDummyRoot* is an alphanumeric data item or literal hosting a boolean value (e.g. "0", "1", "true", "false", "yes", "no", "on" or "off"). If the boolean value is TRUE, then the top level item of Record-Definition is discarded and will not appear in the JSON stream . This parameter can be NULL, in such case, FALSE is assumed.

General rules

1. In Format 1, *type* is set to "application/json" or "application/xml" depending on the Content-type request header field. If no Content-type field is available in the request header, then the type is controlled by the iscobol.rest.default_stream configuration setting.
2. If *content* is an appropriately structured variable with external names and the type is set to "application/json", the method makes a request using the JSON format.

If *content* is an appropriately structured variable with external names but type is not set to "application/json", the method makes a request using the XML format.

getRequestPlain

Returns the HTTP request as plain text.

General format

```
void getRequestPlain( req )
```

Syntax rules

1. *req* is an alphanumeric data item.

General rules

1. This method should be called after a request performed via doPostEx.

getResponseAttachmentIDs

Returns the list of the IDs of the parts in a multipart response.

General format

```
void getResponseAttachmentIDs( ids )
```

Syntax rules

1. *ids* is an alphanumeric data item.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.
2. *ids* is filled with the list of attament IDs. IDs are separated by comma.

getResponseAttachmentAttr

Returns the value of a given attribute of a part in a multipart response.

General format

```
void getResponseAttachmentAttr( id, attrName, attrValue )
```

Syntax rules

1. *id*, *attrName* and *attrValue* are alphanumeric data items.

General rules

1. *id* can be extracted from the list returned by getResponseAttachmentIDs while *attrName* can be extracted from the list returned by getResponseAttachmentAttrNames.
2. *attrValue* receives the attribute value.

getResponseAttachmentAttrNames

Returns the list of the attribute names of the parts in a multipart response.

General format

```
void getResponseAttachmentAttrNames( id, attrNames )
```

Syntax rules

1. *id* and *attrNames* are alphanumeric data items.

General rules

1. *id* can be extracted from the list returned by getResponseAttachmentIDs.
2. *attrNames* is filled with the list of attribute names. Names are separated by comma.

getResponseAttachmentBody

Returns the body of a part in a multipart response.

General format

```
void getResponseAttachmentBody( id, body )
```

Syntax rules

1. *id* and *body* are alphanumeric data items.

General rules

1. *id* can be extracted from the list returned by getResponseAttachmentIDs.
2. *body* receives the body of the part.

Note - if iscobol.http.mtom_enabled (boolean) * is set to true in the configuration, the attachments of a multipart response can be intercepted directly with getResponseXML being stored in fields with either the BASE64BINARY or HEXBINARY clause.

getResponseCode

Returns the numeric HTTP status code from the HTTP response.

General format

```
void getResponseCode( rc )
```

Syntax rules

1. *rc* is a numeric data item.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.

2. Call this method to check if the method call was successful. The response code value for success is 200.

getResponseEx

Returns the HTTP response parsed with JSON or XML rules depending on the Content-type response header field. If no Content-type is available, then it uses the format specified by the iscobol.rest.default_stream configuration setting.

General format

```
void getResponseEx( data )
```

Syntax rules

1. *data* is a level 01 data item for which the IS IDENTIFIED clause has been used.
2. In a JSON response the following escapes are allowed: \b, \f, \n, \r, \t, \" and \\. Any other character preceeded by a backslash is considered invalid and makes the read fail unless you set iscobol.jsonstream.allow_backslash_escaping_any_character (boolean) to true in the configuration.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.

getResponseJSON

Returns the HTTP response parsed with JSON rules.

Format 1

```
void getResponseJSON( json )
```

Format 2

```
void getResponseJSON( json, encoding )
```

Syntax rules

1. *json* is a level 01 data item for which the IS IDENTIFIED clause has been used.
2. *encoding* is a string literal or data item that specifies the character set to be used while parsing the JSON stream. It accepts the same values as the iscobol.encoding * configuration property.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.
2. The following escapes are allowed: \b, \f, \n, \r, \t, \" and \\. Any other character preceeded by a backslash is considered invalid and makes the read fail unless you set iscobol.jsonstream.allow_backslash_escaping_any_character (boolean) to true in the configuration.

getResponseMessage

Returns the text message, if any, from the HTTP response.

General format

```
void getResponseMessage( msg )
```

Syntax rules

1. *msg* is an alphanumeric data item.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.

1. Call this method to check if the request was successful. The response message value for success is "HTTP/1.0 200 OK".

getResponsePlain

Returns the HTTP response as plain text.

General format

```
void getResponsePlain( res )
```

Syntax rules

1. *res* is an alphanumeric data item.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.

getResponseXML

Returns the HTTP response parsed with XML rules.

Format 1

```
void getResponseXML( xml )
```

Format 2

```
void getResponseXML( xml, encoding )
```

Syntax rules

1. *xml* is a level 01 data item for which the IS IDENTIFIED clause has been used.
2. *encoding* is a string literal or data item that specifies the character set to be used while parsing the XML

stream. It accepts the same values as the iscobol.encoding * configuration property.

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.

setAuth

Specify authentication via token (also called Bearer authentication) or via user and password.

Format 1

```
void setAuth( tok )
```

Format 2

```
void setAuth( user, password )
```

Syntax rules

1. *tok* is an alphanumeric data item that contains token authentication.
2. *user* is an alphanumeric data item that contains user name for the authentication.
3. *password* is an alphanumeric data item that contains user name for the authentication.

General rules

1. This method should be called before a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.

setHeaderProperty

Sets HTTP header properties like cookies and charset.

General format

```
void setHeaderProperty( key, value )
```

Syntax rules

1. *key* is an alphanumeric data item that specifies the name of the property to set
2. *value* is an alphanumeric data item that specifies the value for the property

General rules

1. This method should be called before a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.
2. *key* should be specified in lower case otherwise it's not found. For example, in order to get the value of "Content-Type", look for "content-type".

getHeaderProperty

Returns HTTP header properties like cookies and charset.

General format

```
void getHeaderProperty( key, value )
```

Syntax rules

1. *key* is an alphanumeric data item that specifies the name of the property to read
2. *value* is an alphanumeric data item that receives the value of the property

General rules

1. This method should be called after a request performed via one of the following methods: doGet, doPost, doPostEx or doPostMultipart.
2. *key* should be specified in lower case otherwise it's not found. For example, in order to get the value of "Content-Type", look for "content-type".

getMethod

Returns the HTTP request method used for client and server calls.

General format

```
String getMethod( )
```

General rules

1. The returned value is one of the following strings: CONNECT, DELETE, GET, HEAD, OPTIONS, POST, PUT.

setParameter

Sets HTML parameters.

General format

```
void setParameter( name, value )
```

Syntax rules

1. *name* is an alphanumeric data item that specifies the name of the parameter to set
2. *value* is an alphanumeric data item that specifies the value for the parameter

General rules

1. This method should be called before doPost() to prepare parameters to be passed.

saveResponseRaw

Saves the response received from the web server in the specified file.

General format

```
void saveResponseRaw( fileName )
```

Syntax rules

1. *fileName* is an alphanumeric data item that specifies the name of the destination file.

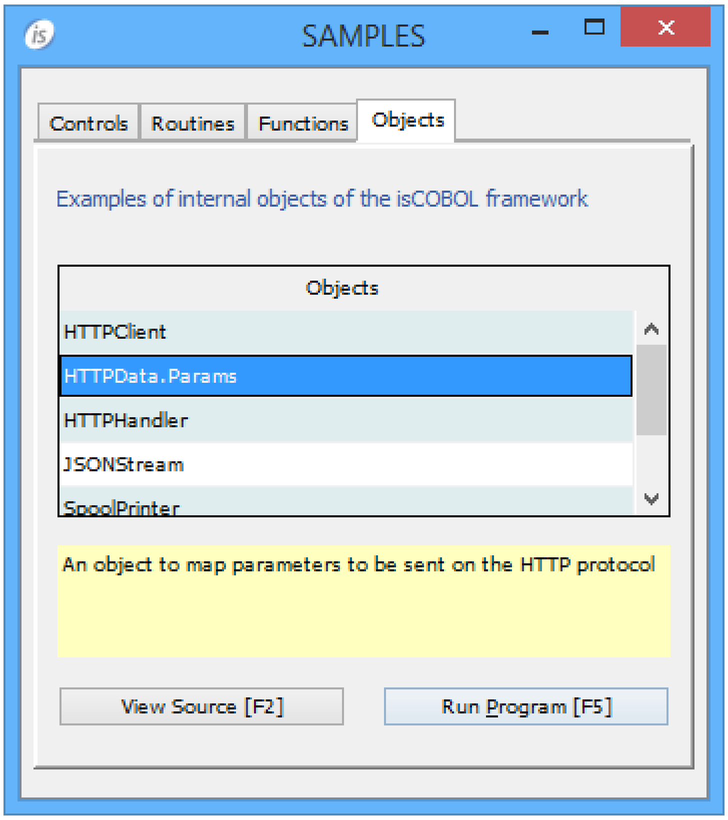
General rules

1. This method should be called when the response is a binary stream (e.g. a JPEG file).

HTTPData.Params class (com.iscobol.rts.HTTPData.Params)

The HTTPData.Params is an internal class that provides a simple way to define HTTP parameters to be passed in doGet and doPost methods.

A sample program can be found in isCOBOL Samples.



Examples of internal objects of the isCOBOL framework

Objects
HTTPClient
HTTPData.Params
HTTPHandler
JSONStream
SpoolPrinter

An object to map parameters to be sent on the HTTP protocol

Constructor

Creates a new instance of the HTTPData.Params class.

General format

```
HTTPData.Params
```

General rules

1. A reference to HTTPData.Params should be defined in the program's Working-Storage Section.

Code example:

```
...
       configuration section.
       repository.
          class http-params as "com.iscobol.rts.HTTPData.Params"
...
       working-storage section.
...
       77 params object reference http-params.
...
       procedure division.
...
       set params to http-params:>new().
...
```

add

Adds an alphanumeric plain-text parameter.

General format

```
void add ( paramName, paramValue )
```

Syntax rules

1. *paramName* is an alphanumeric data item that specifies the parameter name.
2. *paramValue* is an alphanumeric data item that specifies the parameter value.

Example

To define parameters:

```
77 city-zipCode pic x(7) value "26456".
...
  set params = http-param:>new()
                         :>add("get_Zip_Code", city-zipCode).
```

addFile

Adds a disk file to the parameters. This method is useful to implement file upload features. The parameter created by this method is suitable for the doPostMultipart method.

Format 1

```
void addFile ( paramName, fileName )
```

Format 2

```
void addFile ( paramName, fileName, mimeType )
```

Syntax rules

1. *paramName* is an alphanumeric data item that specifies the parameter name.
2. *fileName* is an alphanumeric data item that specifies the disk file name.
3. *mimeType* is an alphanumeric data item that specifies the MIME type.

General Rules

1. *fileName* can be either just the file base name, a relative file path or an absolute file path.
2. In Format 1, the MIME type is automatically set to "application/octet-stream".

Connecting to a SSL-enabled web service

To connect to an SSL web server, you need to add the server's certificate to a local keystore.

Download the server's .cer file, and create a local keystore for it using the following command:

```
keytool -importcert -file <path_to_.cer_file> -keystore <keystore_file.jks> -alias
<keystore_alias>
```

You will be asked to assign a password to the keystore.

At this point you can add the following entries to the isCOBOL configuration (or set them from the program using the SET ENVIRONMENT statement):

```
iscobol.net.ssl.trust_store=/path/to/keystore_file.jks
iscobol.net.ssl.trust_store_password=password
```

Implementing a custom filter for URL match

A filter is an object that is invoked at the preprocessing and postprocessing of a request.

It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc.

The servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet.

In this article we explain how to implement a custom filter for URL match.

A complete and working example is installed with isCOBOL under the folder *samples/eis/webservice/bridge/rest*.

First, we're going to create a class that implements the Filter interface. The filtering logic will be implemented in the *doFilter()* method.

```
       class-id. SusiFilter as "SusiFilter" implements c-filter.

       configuration section.
       repository.
           class j-ioexception as "java.io.IOException"
           class c-filter as "javax.servlet.Filter"
           class c-filter-chain as "javax.servlet.FilterChain"
           class c-filter-config as "javax.servlet.FilterConfig"
           class c-ServletException as "javax.servlet.ServletException"
           class c-ServletRequest as "javax.servlet.ServletRequest"
           class c-ServletResponse as "javax.servlet.ServletResponse"
           .
       id division.
       object.
       data division.
       working-storage section.

       procedure division.

       id division.
       method-id. init as "init".
       linkage section.
       77  cfg object reference c-filter-config.
       procedure division using cfg raising c-ServletException.
       main.
       end method.

       id division.
       method-id. c-destroy as "destroy".
       procedure division.
       main.
       end method.

       id division.
       method-id. doFilter as "doFilter".
       working-storage section.
       linkage section.
       77  request            object reference c-ServletRequest.
       77  response           object reference c-ServletResponse.
       77  f-chain            object reference c-filter-chain.

       procedure division using request response f-chain
                        raising c-ServletException j-IOException.
       MAIN.
      *>    << filtering logic here >>
       end method.

       end object.
```

Filter logic

Refer to the installed sample for the COBOL code to be written in the *doFilter()* method.

Here is an explanation of the logic implemented in the sample.

Once the REST API has been planned, it needs to be described and mapped so that the isCOBOL runtime can understand it.

To do so, 2 string arrays are used

- tableReq: contains the patterns that define the application's REST API.
- tableReqIsc: contain the translation from the REST API to the standard isCOBOL notation using URL parameters.

Below are 3 samples:

```
move "GET/.+/songs/last"                   to tableReq(6)
move "/servlet/SONGS?Operation=L"          to tableReqIsc(6)

move "GET/.+/songs/print"                  to tableReq(7)
move "/servlet/SONGS?Operation=p"          to tableReqIsc(7)

move "GET/.+/songs/next/[0-9]+"            to tableReq(4)
move "/servlet/SONGS?Operation=N&id={4}"   to tableReqIsc(4)
```

The matching process is based on regular expression checking.

The tableReq array holds a list of regular expression strings that describe REST URLs and parameters.

The string is composed by three logical parts:

1. The HTTP method (GET / PUT / POST / DELETE)
2. The Webapp name
3. The resource name and parameters needed for the REST call

In this sample the webapp name is matched using the regular expression ".+/" which accepts any string, as the only resource used is SONGS.

The parameters are expressed using a combination of fixed strings and the following regular expression patterns:

1. [0-9] used to accept numeric parameter, of any length
2. [a-zA-Z] used to accpet to accept alphabetic parameter, of any length
3. .+ used to accept any kind of charactrs, of any length

The tableReqIsc array holds the URL needed by the isCOBOL runtime to perform the REST request. This string may contain placeholders, identified by a integer number enclosed in brackets, which represent the original request ordinal parameter, and will be explained below.

REST paramters are usually embedded in the URL (such as /songs/next/3) to request the song following the one with id 3. In this case, the 3 is a variable parameter, while "songs" and "next" are fixed parameters.

The tableReq entry to match this URL is: "GET/.+/songs/next/[0-9]+"

The matching process performs the following steps:

1. It determines the tableReq entry that matches the request URL
2. If none is found, a HTTP error 400 BAD REQUEST is returned
3. If one is found, the incoming URL request is split in segments on the "/" character
4. The tableReqIsc arrays is used to rearrange the URL parameters in the resulting isCOBOL URL
5. The request if forwarded to the resulting isCOBOL URL

For example, if the URL /songs/next/10 is requested, it matches the "GET/.+/songs/next/[0-9]+" tableReq entry, and the corresponding tableReqIsc entry is "/servlet/SONGS?Operation=N&id={4}".

The REST URL is split into the following 0-based array of parts:

- [0] - the request method, in this case "GET"
- [1] - the wepapp name, which we will not need
- [2] - the resource name, "songs"
- [3] - the parameter "next"
- [4] - the parameter "10"

The tableReqIsc entry only indicates to extract parameter number 4, in this case "10" and the corresponding isCOBOL URL is then composed, yelding the final URL "/servlet/SONGS?Operation=N&id=10".

The request is redirected to this URL, and the isCOBOL program is invoked.

Filter activation

Once the filter class is ready, your can edit the web.xml descriptor and add the following filter mapping:

```
<filter>
      <filter-name>isCOBOL filter</filter-name>
      <filter-class>SusiFilter</filter-class>
</filter>
<filter-mapping>
      <filter-name>isCOBOL filter</filter-name>
      <url-pattern>/*</url-pattern>
</filter-mapping>
```

From now on, all the requests received by the webapp will be processed by SusiFilter.

Useful definitions

User Agent / Client

The program that is used to request information from a server. This program is frequently a web browser, but it could be any program on the user's machine.

HTTP

Hypertext Transport Protocol, a standard encoding scheme used to transmit requests to web servers and receive responses from web servers. HTTPS is a secure version of HTTP.

Request

An HTTP packet that contains a command issued by the user agent. A request may simply GET a file from a web server, PUT a file to the web server, DELETE a file from the web server, or may POST data (such as a form) to the server, or it may cause a program to be run on the server. GET and POST are by far the most frequently used commands.

URL

Uniform Resource Locator, the location of a resource on the internet. A URL consists of a scheme (in this context, HTTP or HTTPS), the name of a machine, and a path to a file. For example, https://www.veryant.com/eis/index.html specifies the file called index.html from directories on a server machine veryant.com using the HTTP scheme. When this is typed into a web browser, the browser issues a HTTP GET request on this file.

REST

REST (Representational State Transfer) is an architectural style for distributed hypermedia systems and can be used to implement web services. While there is not a formal standard like SOAP, it is based on the four principle HTTP request types (GET, PUT, POST and DELETE), and URLs. In a REST architecture, a request payload be in any format desired, including XML or JSON.

Web Server

A program that runs on a server and listens for HTTP requests. When a request is received, the web server processes the request or sends it on to another program (such as J2EE Container like Tomcat) for processing.

Web Service (or WS)

A software system designed to support interoperable machine-to-machine interaction over a network

Servlet Container

A process that handles the execution of COBOL Servlets, turning them into a web page that the web server can deliver back to the client.

Response

A HTTP packet that contains the response to the request. The response may be text, to be displayed in a web browser, or data encapsulated for consumption by the requesting program.

Session

Requests are stateless, that is, the web server processes each request as if it had never received a previous request from the same user agent. A session is a BIS concept that allows sequential requests from the same user agent to be grouped together and preserves state information across requests on the server.

AJAX

Ajax (an acronym for Asynchronous JavaScript and XML) is a group of interrelated web development techniques used on the client-side to create asynchronous web applications

JS

JavaScript source code, or based on JavaScript source code

SOAP

(from http://www.w3.org/TR/2007/REC-soap12-part1-20070427): a SOAP message is specified as an XML infoset whose comment, element, attribute, namespace and character information items are able to be serialized as XML 1.0. Note, requiring that the specified information items in SOAP message infosets be serializable as XML 1.0 does NOT require that they be serialized using XML 1.0. A SOAP message Infoset consists of a document information item with exactly one member in its [children] property, which MUST be the SOAP Envelope element information item (see 5.1 SOAP Envelope). This element information item is also the value of the [document element] property. The [notations] and [unparsed entities] properties are both empty. The Infoset Recommendation [XML InfoSet] allows for content not directly serializable using XML; for example, the character #x0 is not prohibited in the Infoset, but is disallowed in XML. The XML Infoset of a SOAP Message MUST correspond to an XML 1.0 serialization [XML 1.0].

WSDL

(from https://www.w3.org/TR/2001/NOTE-wsdl-20010315): A WSDL document defines services as collections of network endpoints, or ports. In WSDL, the abstract definition of endpoints and messages is separated from their concrete network deployment or data format bindings. This allows the reuse of abstract definitions: messages, which are abstract descriptions of the data being exchanged, and port types which are abstract collections of operations. The concrete protocol and data format specifications for a particular port type constitutes a reusable binding. A port is defined by associating a network address with a reusable binding, and a collection of ports define a service.