

Build a RESTful service on CICS with PHP

Skill Level: Intermediate

[Robin Fernandes \(robin_fernandes@uk.ibm.com\)](mailto:robin_fernandes@uk.ibm.com)

Software Developer
IBM

[Jonathan Lawrence \(jlawrence@uk.ibm.com\)](mailto:jlawrence@uk.ibm.com)

Software Developer
IBM

21 Apr 2009

CICS® Transaction Server® (TS) is a powerful transaction manager designed for rapid, high-volume processing. SupportPac CA1S uses technology from IBM WebSphere® sMash to enhance CICS TS with PHP scripting capabilities and Representational state transfer (REST)-related features. This tutorial shows how you can use PHP to quickly and easily work with CICS programs and expose them on the Web. If you are a PHP developer, find out how you can use your skills to interact with enterprise assets in CICS; if you are a CICS developer, see how PHP provides a simple and agile way to manipulate your existing resources.

Section 1. Before you start

This tutorial shows how you can use CICS SupportPac CA1S to quickly expose CICS COMMAREA programs on the Web with PHP—a simple and powerful language that is ideally suited to rapid Web development. You will create a Web service that uses REST and JavaScript Object Notation (JSON), ensuring that it can easily be consumed by a variety of clients, such as Asynchronous JavaScript and XML (Ajax) front ends, other Web services, and mash-ups.

The example CICS COMMAREA program provided with this tutorial (see [Download](#)) is a simple library application written in COBOL. It holds a list of books in a VSAM file to which entries can be added or removed, and allows books to be marked as borrowed or returned.

In this tutorial you will:

- Set up the library program on your CICS system.
- Learn how to invoke CICS COMMAREA programs from PHP.
- Learn about the features in CA1S that simplify the creation of RESTful Web services.
- Expose the library program as a RESTful Web service

Prerequisites

To get the most out of this tutorial, you should have a basic understanding of the PHP language and some familiarity with CICS administration or CICS program development. To read up on CICS, visit the [CICS TS 3.2 Information Center](#). You may also want to read through the introduction section of the CA1S user guide, which briefly outlines the purpose of CA1S and the principles of REST. (See [Resources](#) for links to these guides and documentation.)

CICS environment

To complete the tutorial, you will need a CICS environment that meets the prerequisites listed on the [CA1S download page](#).

Download and install the CA1S SupportPac following the instructions in the [CA1S user guide](#). Verify your installation by invoking the HelloCICS.php script as described in the guide.

Tools

- Any text editor is sufficient to write PHP scripts. You can use an IDE such as Eclipse PDT.
- You will need a mechanism to transfer PHP scripts from your workstation to your CICS server, such as an FTP client or the Target Management plug-in for Eclipse.
- To test your RESTful Web services, you will need a simple REST client that can issue GET, POST, PUT, and DELETE HTTP requests, such as the Poster add-on for Mozilla Firefox.

Downloads for all tools described can be found in [Resources](#).

Section 2. Setting up the Library application in CICS

This section outlines the administrative tasks required to install the sample LIBRARY application in CICS, and describes the use of the COMMAREA interface to the program from a client. This interface is needed to write a PHP script that accesses the program.

Steps to set up the Library application

The Library application requires a VSAM file to be created and populated, and a COBOL program to be compiled. CICS resource definitions for the file and program are also required. These steps are outlined in more detail below. Some knowledge of CICS configuration procedures is assumed.

No mechanism is provided to invoke the LIBRARY program except through PHP as described later in this article, although one could be developed. Therefore, there is limited opportunity to verify the correct installation of the File and Program, and for this reason, the setup instructions suggest that the file be opened and the program loaded manually using `CEMT` in order to check these definitions.

Creating and defining the VSAM file

The example library program requires a VSAM KSDS to hold the contents of the library catalog. The `library/jcl/LIBFILE` file in the supplied code archive contains sample JCL to delete and recreate the required dataset, and populate it with suitable initial data. This JCL needs to be transferred to a suitable location in your z/OS® system, and modified as follows:

- Replace the JOB card with one suitable for your environment.
- Replace all instances of the dataset name (`P8BUILD.CICS650.LIBRARY.LIBFILE`) with a suitable dataset name for your system. The chosen name is required later for the CICS FILE definition.
- Optionally, edit the initial data for the library, taking care to preserve consistency and integrity (see note below).

The following steps complete the creation and definition of the VSAM file:

1. Submit the LIBFILE JCL as modified above, and check for successful execution.

2. Log into CICS and create a CICS FILE definition using CEDA (or other mechanism) with the following attributes:

Table 1. Attributes to use when creating the VSAM FILE definition

Attribute	Value
FILE	LIBFILE
GROUP	LIBRARY
DSName	<As specified in LIBFILE JCL>
RECORD FORMAT	F
ADD	Yes
BROWSE	Yes
DELETE	Yes
READ	Yes
UPDATE	Yes

3. Install the LIBFILE definition and check for success.
4. Optionally, using CEMT or otherwise, open LIBFILE and check that it opens successfully:

```
CEMT SET FILE(LIBFILE) OPEN
```

This step is suggested to check that the file has been created and defined correctly, because limited verification of the setup is possible.

Compiling and defining the COBOL program

The source code for the LIBRARY application is in the file library/cobol/LIBRARY, and the sample JCL to compile it is in the file library/jcl/LIBCOBOL, both in the sample code archive file provided with this article. These files should be transferred to suitable locations in your z/OS system, and the LIBCOBOL JCL should be modified as described below. The supplied version is as follows:

Listing 1. The JCL to compile the LIBRARY program

```
//LIBCOBOL JOB USER=P8BUILD,CLASS=A,MSGCLASS=A,NOTIFY=&SYSUID
// JCLLIB ORDER=CTS320.CICS650.SDFHPROC
//COMPCOB EXEC DFHZITCL,
// PROGLIB=P8BUILD.CICS650.LOAD,
// LNGPRFX=PP.COBOL390.V340,
// CBLPARM=( 'NODYNAM,LIB,ADATA,RENT', COMPILER OPTIONS
```

```
//          'CICS(''COBOL2,SP'' )',          TRANSLATOR OPTIONS
//          INDEX=CTS320.CICS650,
//          LIBPRFX=PP.ADLE370.ZOS190,
//          DSCTLIB=P8BUILD.CICS650.COPY
//COBOL.SYSIN DD DSN=P8BUILD.CICS650.COBO(LIBRARY),DISP=SHR
//COBOL.SYSADATA DD DSN=P8BUILD.CICS650.COBADATA(LIBRARY),
//          DISP=SHR
//LKED.SYSIN DD *
//          NAME LIBRARY(R)
/*
```

The required changes are as follows:

- Provide a suitable JOB card.
- Change all dataset names and prefixes as appropriate for your z/OS and CICS installation.
- Ensure that the COBOL.SYSADATA DD definition points to a member of a PDS with variable length records, which is suitable to contain the ADATA output. Refer to the COBOL compiler documentation for more details.

The following steps complete the compilation and installation of the LIBRARY program:

1. Submit the LIBCOBOL JCL as modified above, and check for successful execution.
2. Log into CICS and create a CICS PROGRAM definition using CEDA (or other mechanism) with the following attributes:

Table 2. CICS PROGRAM attributes

Attribute	Value
PROGRAM	LIBRARY
GROUP	LIBRARY
Language	COBOL

3. Install the LIBRARY program definition and check for success.
4. Optionally, using CEMT or otherwise, NEWCOPY the LIBRARY program, and check that it loads:

```
CEMT SET PROGRAM(LIBRARY) NEWCOPY
```

This step is suggested to check that the program has been compiled and defined correctly, because limited verification of the setup is possible.

How to use the COBOL Library application

The convention for a CICS COMMAREA program is that when it is invoked, an area of storage is passed to the program containing input data, sometimes including an indication of the operation that is to be performed. On completion of processing, the storage area is overwritten with output data and response codes from the operation. The format of the COMMAREA storage area is often defined in a COBOL copybook, which is included in the target program and its callers.

To use such a CICS program, you need to know not only the format of the COMMAREA expected by the program for input and output, but also the supported operations and the expected input and output data and response codes. In CA1S, the task of building and interpreting the data structure of the COMMAREA is performed by the JZOS-generated record classes. However, the PHP programmer is responsible for setting the required fields on input and extracting the appropriate data on output, using the JZOS-generated APIs.

The supplied example Library application follows this standard pattern, and therefore, the PHP programmer needs details of the use of the COMMAREA interface in addition to the generated JZOS record classes, in order to invoke the Library program correctly from a PHP script.

Library COMMAREA format

The COMMAREA format used by the library program is defined by the following COBOL data definition:

Listing 2. The library program's COBOL data definition

```
01 DFHCOMMAREA.
*   LIBRARY COMMAREA structure
03 LIB-REQUEST-TYPE          PIC X(6).
03 LIB-RETURN-CODE          PIC 9(2).
03 LIB-ITEM-COUNT           PIC 9(4).
03 LIB-BOOK-ITEM OCCURS 24 TIMES.
    05 BOOK-ITEM-REF        PIC 9(4).
    05 BOOK-TITLE           PIC X(20).
    05 BOOK-AUTHOR          PIC X(20).
    05 BOOK-LOAN-STATUS     PIC 9(2).
        88 BOOK-ONLOAN VALUE 1.
        88 BOOK-UNLENT VALUE 0.
    05 BOOK-BORROWER        PIC X(20).
    05 FILLER                PIC X(14).
```

The JZOS record generator tool, described later, converts the definition shown in Listing 2 into APIs that can be used from PHP to get and set fields in the COMMAREA. There is an intuitive mapping between the field names used in the COBOL and the PHP methods so that these can be related to each other.

Library program use

Knowledge of the COMMAREA format expected by the target CICS program alone is insufficient to allow the program to be invoked correctly. It is also necessary to have usage information for the program, such as what operations are supported by the program, how this is specified in the COMMAREA, and the other inputs and outputs for each operation, together with possible outcomes and responses. This usage information for the library program is given in Table 3 for reference.

Because sample PHP scripts to invoke the library application are provided with this article, you do not need to use this information for the initial setup, but you do need it to understand, modify, or re-implement the scripts.

LIB-REQUEST-TYPE is a 6-character text field that determines the operation to be performed by the Library program, and must be set to one of the following values before calling the program:

Table 3. The library program's LIB-REQUEST-TYPE field

LIB-REQUEST-TYPE	Library operation
LIST	Return a list of all books in the library in the array LIB-BOOK-ITEM. This may contain up to 24 elements, which is the maximum capacity of the library. Each item contains full details of the associated book. The number of books present is returned in LIB-ITEM-COUNT.
QUERY	Return details of specified book in the first element, LIB-BOOK-ITEM(1). The index (reference) for the requested book is input in BOOK-ITEM-REF(1).
ADD	Add a new book to the library; details supplied in LIB-BOOK-ITEM(1). Provided there is space in the library (max books = 24), the index for the new book is returned in BOOK-ITEM-REF(1).
DELETE	Delete the book identified by BOOK-ITEM-REF(1) from the library, if it exists.
BORROW	Mark the book specified by BOOK-ITEM-REF(1) as being borrowed, and add the supplied borrower name (BOOK-BORROWER(1)) to the book record.

RETURN	Mark the specified book as not borrowed and delete the borrower name from the book record. The book to be returned is identified by <code>BOOK-ITEM-REF(1)</code> .
---------------	---

On return, the Library program sets `LIB-RETURN-CODE` to a two-digit numeric value depending on the outcome of the requested operation:

Table 4. The library program's return codes

<code>LIB-RETURN-CODE</code>	Meaning		
Field name	Value	Operation	Condition
<code>LIBRARY-OK</code>	0	All	Successful completion.
<code>LIBRARY-NOTFOUND</code>	1	All except LIST, ADD	Specified book not found in library.
<code>LIBRARY-ONLOAN</code>	2	BORROW	Book is already out on loan.
<code>LIBRARY-DUPLICATE</code>	3	Not used	Should not occur.
<code>LIBRARY-INVREQ</code>	4	All	Invalid user request.
<code>LIBRARY-FULL</code>	5	ADD	Library is full (maximum 24 books!).
<code>LIBRARY-ERROR</code>	99	All	Unexpected processing error.

The available request types and return codes are defined in a level-01 data section (`LIBRARY-CONSTANTS`) in the library program, which means that they can be made available to a PHP script using the JZOS record generator in a similar way to the `COMMAREA` data section.

In general, the PHP programmer will need information equivalent to the information in Table 4 for each CICS program to be called from PHP, and this will typically be provided in a suitable form by the CICS programmer responsible for maintaining the program.

Section 3. Invoking the Library program from PHP

Making the COMMAREA available to PHP

After the COMMAREA is defined in the COBOL program and the ADATA has been generated, the next step is to create Java™ classes that represent the COMMAREA. These classes are used to make the COMMAREA accessible from PHP scripts.

You will only need to generate the Java classes once: as soon as the classes are available to CA1S, any number of scripts may be written to use them to interact with the COBOL program. However, if the COBOL program is modified such that the shape of the COMMAREA changes, you will need to regenerate the ADATA and the Java classes, and adapt the PHP scripts as required.

The following steps require a Java SDK and assume that the `java` and `javac` commands are available on the system PATH. You execute them on your workstation or directly on the CICS system.

1. Generate the source of the COMMAREA classes

```
java -cp jzos_recgen.jar
com.ibm.jzos.recordgen.cobol.RecordClassGenerator
genCache=false
adataFile=LIBRARY symbol=DFHCOMMAREA class=Library_Commarea
package=library outputDir=.
```

Let's describe the purpose of each part of the above command:

```
java -cp jzos_recgen.jar com.ibm.jzos.recordgen.cobol.RecordClassGenerator
```

This invokes the JZOS Record Generator, which is a Java program contained in the file `jzos_recgen.jar`. This file is included in CA1S, and the most recent version is available on the IBM JZOS Batch Toolkit for z/OS SDKs site at [alphaWorks](#) (see [Resources](#)).

Why do I get a Java exception when attempting to run the JZOS record generator?

Before retrieving the ADATA using FTP, remember to run the

```
command 'quote site rdw' and 'bin'.
```

For more information, see the section "Running the COBOL RecordClassGenerator" of the JZOS documentation, which is included in the CA1S package as "JZOS Cobol Record Generator Users Guide.pdf".

```
genCache=false
```

This option ensures that the generated Java code does not attempt to cache the value of COMMAREA fields. **This option is required** for CA1S to interact correctly with the generated classes.

```
adataFile=LIBRARY
```

This specifies the path to the ADATA file obtained in the previous section.

```
symbol=DFHCOMMAREA
```

This specifies the name of the COMMAREA for which to generate the class.

```
class=Library_Commarea
```

This specifies the name of the resulting Java class.

```
package=library
```

This specifies the package of the resulting Java class.

```
outputDir=.
```

This specifies the path to which the Java class will be written on the file system.

2. Generate the Java source for the class representing COBOL constants

In the sample library application, the constants used in the COBOL program (for the operation names and response codes) are defined in a data structure in the WORKING-STORAGE section. This means that a Java class representing these

constants can be generated and used from PHP in a similar way to the class representing the COMMAREA, in this case to obtain the values of the defined constants where needed.

```
java -cp jzos_recgen.jar com.ibm.jzos.recordgen.cobol.RecordClassGenerator genCache=false
adataFile=LIBRARY symbol=LIBRARY-CONSTANTS class=Library_Constants package=library
outputDir=.
```

The advantage of this approach is that PHP reflection can be used to examine the available constant names, and the PHP programmer need not refer directly to the values defined in the COBOL. Keep in mind that this can be done for the LIBRARY application because of the way the constants have been defined, and may not always be possible; for example, where hard-coded values are embedded in the COBOL source.

In the PHP scripts provided with this article, a mixed approach is adopted, where some of these constants are used in their literal forms (for example, the operation names), and some are extracted using the generated Java class (the numeric response codes).

3. Compile the .java source files to create Java class files

The generated Java source will be in the `outputDir` specified above, in subdirectory `library` (which corresponds to the package name). Use `javac` to compile the source files as follows:

```
javac -cp jzos_recgen.jar library/*
```

4. Make the class files available to CA1S

The library directory now contains the compiled classes. For CA1S to be able to use them, they must be available on the Java classpath. If you generated the classes on your workstation, you will need to transfer the directory to the server (for example, using FTP).

The Java classpath is determined by the `CLASSPATH_SUFFIX` attribute of the `JVMPROFILE` used by CA1S. The default `JVMPROFILE` in CA1S (named `CA1SJVM`) is already configured to include the directory `ca1s/work/classes/` in the `CLASSPATH_SUFFIX`:

```
CLASSPATH_SUFFIX=/u/p8build/cals/config/ini:\
/u/p8build/cals/p8/jars/p8api.jar:\
/u/p8build/cals/p8/jars/p8.jar:\
/u/p8build/cals/p8/jars/p8cics.jar:\
/u/p8build/cals/work/classes:\
```

```
/usr/lpp/db2910/classes/db2jcc.jar:\
/usr/lpp/db2910/classes/db2jcc_javax.jar:\
/usr/lpp/db2910/classes/db2jcc_license_cisuz.jar
```

Therefore, if you are using JVMPROFILE CA1SJVM, you may copy the library directory to ca1s/work/classes/. Alternatively, you may change your CLASSPATH_SUFFIX to include a directory or JAR file that contains the library directory.

Note that the directory structure representing the package name must be preserved under the classpath location. So, if you are using ca1s/work/classes/, copy the library directory itself such that the classes are under ca1s/work/classes/library (and not directly under ca1s/work/classes/).

Finally, phase out your JVMs to ensure the new classes are picked up by CA1S.

Accessing the program from PHP code

The code

The invocation of a COMMAREA program consists of 3 steps:

1. Preparing the COMMAREA before linking to the program.
2. Linking to the program.
3. Retrieving the result of the link from the COMMAREA.

To illustrate, Listing 3 shows a script that invokes the library program to obtain the list of books, then prints them out:

Listing 3. Invoking a CICS Commarea program from PHP

```
<?php
// Step 1: create and prepare the COMMAREA instance
java_import('library.Library_Commarea');
$COMMAREA = new Library_Commarea();
$COMMAREA->setLibRequestType('LIST');

// Step 2: create the program instance and invoke it using the COMMAREA
$program = new CICSProgram('LIBRARY');
try {
    $program->link($COMMAREA);
} catch (CICSException $e) {
    echo 'Error: ' . $e->getMessage();
    return;
}
```

```
// Step 3: retrieve the result of the link from the COMMAREA
$totalBooks = $COMMAREA->getLibItemCount();
echo "Total number of books: $totalBooks <br/>";

for ($i=0; $i<$totalBooks; $i++) {
    $book = $COMMAREA->getLibBookItem($i);
    $title = $book->getBookTitle();
    $author = $book->getBookAuthor();
    echo "Book $i is '$title' by '$author'. <br/>";
}
?>
```

Why do I just see garbage characters when I access a script in my browser?

This can happen if the script is encoded in the wrong code-page. By default, the runtime for PHP in CA1S is configured to expect scripts in the UTF-8 encoding. Therefore, if you transfer scripts between a Windows® or Linux® workstation and your CICS server, be sure to set "binary" mode, so that the scripts are not converted to an EBCDIC code-page during the transfer. For more information on encoding concerns and configuration options in CA1S, see the [CA1S documentation](#).

This script is provided in the sample code download as `library/scripts/library.php`. If you transfer this script to your CICS system to `ca1s/work/scripts/library.php` and access it in your browser, you should see a list of books like:

```
Total number of books: 18
Book 0 is 'PHP for Beginners ' by 'Rob
Nicholson '.
Book 1 is 'Project Management ' by 'A N
IBMer '.
Book 2 is 'Easy Z Specification' by
'Jonathan '.
Book 3 is 'REST Protocol Design' by 'Zoe,
Ant & Rob '.
etc...
```

Let's examine the three steps in the code in detail.

Preparing the COMMAREA

```
java_import('library.Library_Commarea');
```

The function `java_import()`, which is built in to the runtime for PHP in CA1S, loads up a Java class and makes it available for use within PHP. In the code above, we load the Java class representing the library COMMAREA class that we generated previously.

```
$COMMAREA = new Library_Commarea();
```

Then, we create an instance of this class. This instance will be used as a container for the input and output data of the invocation.

```
$COMMAREA->setLibRequestType('LIST');
```

Finally, on the third line, we set some input data on the COMMAREA. The method `setLibRequestType` is specific to the Java class representing the COMMAREA of the COBOL program used in this tutorial: it defines the operation we want to perform against the library, in this case obtain a LIST of all the books. See [Investigating the COMMAREA using reflection](#) to learn how you can use PHP to discover all the methods on the Java class.

Linking to the program

```
$program = new CICSProgram('LIBRARY');
```

First, we create an instance of the built-in class `CICSProgram`, which represents the CICS program with which we want to interact. The name of the CICS program is specified as the constructor argument, in this case "LIBRARY." If the program were to be renamed, this argument would need to be changed.

Why do I see "com.ibm.cics.server.InvalidProgramIdException: CICS PGMIDERR Condition" instead of the expected output?

Double-check that the argument passed to the `CICSProgram` constructor represents the name of the COBOL program installed on your CICS system.

```
$program->link($COMMAREA);
```

The `link` method on the `CICSProgram` class triggers the execution of the CICS program. If a single COMMAREA argument is supplied, as is the case here, it will be used by the CICS program as a container for both input and output data. You may also supply two separate COMMAREA arguments, in which case the first argument will be expected to contain input data, and any output data will be written to the second. It is also possible to supply no arguments, which is useful in cases where the CICS program has no input and output.

```
try / catch block
```

If the link fails or the linked program terminates abnormally, the link method will throw a `CICSEException`. The `getMessage()` method on the exception object reveals details of the failure, including the type of the underlying Java exception. For example, if an invalid program name is specified as the argument to the `CICSProgram` constructor, the code above would print:

```
Error: com.ibm.cics.server.InvalidProgramIdException: CICS PGMIDERR Condition
```

Retrieving data from the COMMAREA

```
$totalBooks = $COMMAREA->getLibItemCount();
```

After `link()` successfully returns, the `COMMAREA` contains the output of the invocation. In step 3 of the PHP script above, we use various methods specific to the `Library_Commarea` class to obtain the total number of books, then to iterate over the list of books, printing the title and author of each one.

Investigating the Commarea using PHP reflection

Setting input data on a `COMMAREA` before a link and retrieving output data from it afterwards requires knowledge of the setter and getter methods on the `COMMAREA` class. If you don't have a list of these methods at hand, it is simple to obtain one with PHP reflection.

For example, the following script uses the PHP reflection classes (see [Resources](#)) to obtain the list of methods available of the class `Library_Commarea`:

Listing 4. Using reflection, part 1

```
<?php
java_import('library.Library_Commarea');
$COMMAREA = new Library_Commarea();
$rc = new ReflectionObject($COMMAREA);
foreach ($rc->getMethods() as $method) {
    echo $method->getName() . '<br/>';
}
?>
```

The output of the script in Listing 4 is:

```
Library_Commarea
__toString
setLibItemCount
getLibReturnCode
setLibRequestType
setLibReturnCode
```

```
getByteBuffer  
getLibRequestType  
getLibBookItem  
getLibItemCount
```

Aside from method `Library_Commarea()` (which is the constructor) and `getByteBuffer()` (which provides access to the raw byte array holding the COMMAREA data), all the other methods are getters and setters for COMMAREA fields. Those used in the initial script are marked in bold.

Some getters return complex data structures, such as the `getLibBookItem()` method above, which returns an object representing a book. The same technique can be used on these data structures to discover which fields can be accessed. For example, here we use the PHP function `get_class_methods()` (see [Resources](#)) to establish which fields can be accessed on a book:

Listing 5. Using reflection, part 2

```
<?php  
// Step 1: create and prepare the COMMAREA instance  
java_import('library.Library_Commarea');  
$COMMAREA = new Library_Commarea();  
$COMMAREA->setLibRequestType('LIST');  
  
// Step 2: create the program instance and invoke the program using the COMMAREA  
$program = new CICSProgram('LIBRARY');  
$program->link($COMMAREA);  
  
// Investigate the fields available on a book object  
$book = $COMMAREA->getLibBookItem(0);  
print_r(get_class_methods($book));  
?>
```

The output of the script in Listing 5 is:

```
Array  
(  
    [0] => LibBookItem  
    [1] => getBookTitle  
    [2] => setBookItemRef  
    [3] => isBookOnloan  
    [4] => getBookAuthor  
    [5] => isBookUnlent  
    [6] => getByteBuffer  
    [7] => setBookAuthor  
    [8] => setBookBorrower  
    [9] => getBookLoanStatus  
    [10] => getByteBufferOffset  
    [11] => getBookBorrower  
    [12] => __toString  
    [13] => getFiller_1  
    [14] => getBookItemRef  
    [15] => setBookTitle  
    [16] => setFiller_1  
    [17] => setBookLoanStatus  
)
```

Troubleshooting the classpath with phpinfo()

If you see warnings or errors suggesting that the COMMAREA class cannot be found, check your Java classpath to ensure it includes a directory or JAR file containing the library directory (which contains the generated classes).

This can be done directly from php code using the PHP function `phpinfo()`:

```
<?php
phpinfo();
?>
```

Accessing that script in a browser will display lots of information, including the full classpath:

Figure 1. Checking your Java classpath with phpinfo()

CICS_HOME	/cics/cics650/
STDERR	logs/dfhjvmerr-generate
LIBPATH_SUFFIX	/usr/lpp/db2910/lib:/u/p8build/ca1s/p8/lib
CLASSPATH	-Djava.class.path=/cics/cics650/lib/dfjcicsras.jar:/cics/cics650/lib/ras.jar:/cics/cics650/lib/dfjorb.jar:/cics/cics650/lib/dfjcont.jar:/cics/cics650/lib/dfjcdmn.jar:/cics/cics650/lib/dfjjts.jar:/cics/cics650/lib/dfjname.jar:/cics/cics650/lib/websphere.jar:/cics/cics650/lib/dfjejbdd.jar:/cics/cics650/lib/dfjoutput.jar:/cics/cics650/lib/wsd/dfjwsdl/woden.jar:/cics/cics650/lib/wsd/NamespacesCon/xmlSchema.jar:/java/java50_bit31_sr8/J5.0/standard/ejb/2_0/ejb20.jar:/jta/1_0_1/jta-spec1_0_1.jar:/java/java50_bit31_sr8/J5.0/standard/jca/c/p8build/ca1s/p8/jars/p8api.jar:/u/p8build/ca1s/p8/jars/p8.jar:/u/p8build/ca1s/work/examples/jars/catalog.jar:/u/p8build/ca1s/work/classes:/usr/db2910/classes/db2jcc_javax.jar:/usr/lpp/db2910/classes/db2jcc_licer
JAVA_HOME	/java/java50_bit31_sr8/J5.0/

classpath.png

More generally, `phpinfo()` is a useful tool for investigating many aspects of the PHP environment.

Section 4. REST basics in CA1S

RESTful event handlers in SupportPac CA1S

Overview

WebSphere sMash introduces a set of conventions that simplify the creation of RESTful Web services (for more information, see the WebSphere sMash documentation in [Resources](#)). SupportPac CA1S adopts a subset of these conventions. They are described in detail in the CA1S documentation (see [Resources](#)), but to summarise:

- CA1S can be configured to treat requests to certain paths as RESTful. In a default CA1S installation, this applies to paths under /ca1s/resources
- When a request on such a path is processed, information about the targeted resource is derived from the URI as follows:

```
http://<host>/cals/resources/<resourceName>[ /<resourceID>[ /<...more/resource/info...> ] ]
```

- A resource handler script corresponding to the resource name is looked up. The PHP engine will execute that script, then attempt to invoke a specific method within the script depending on the HTTP method of the request and whether the URI contains an identifier after the resource name.

Table 5 shows the mapping from the HTTP request type to the method that will be invoked, assuming the resource is named 'book':

Table 5. RESTful events handler methods

Request URI	HTTP Method	Handler method invoked in ca1s/work/resources/book.php
Collection URI, e.g. http://<host>/cals/resources/book	GET	book::onList()
	POST	book::onCreate()
	PUT	book::onPutCollection()
	DELETE	book::onDeleteCollection()
Member URI, e.g. http://<host>/cals/resources/book/10	GET	book::onRetrieve()

	POST	book::onPostMember()
	PUT	book::onUpdate()
	DELETE	book::onDelete()

Resource handler scripts do not need to implement all the methods shown in Table 5. If a method is not found for a given event, no error is raised. The most commonly implemented event handlers are for the events that correspond to List, Create, Retrieve, Update, and Delete ("LCRUD") operations on the resource, as highlighted in bold in Table 5.

Example

The example shown in Listing 6 prints a different string for each event:

Listing 6. A skeleton resource handler script

```
<?php
class book {
    function onList() {
        echo 'LIST all books!';
    }

    function onCreate() {
        echo 'CREATE a book!';
    }

    function onRetrieve() {
        echo 'RETRIEVE a book!';
    }

    function onUpdate() {
        echo 'UPDATE a book!';
    }

    function onDelete() {
        echo 'DELETE a book!';
    }
}
?>
```

To try it out, copy the script above to your CICS system in `ca1s/work/resources/book.php` (note that the name of the class declared in the script must match the script file name). Then, access it with different HTTP methods, with and without an identifier on the URI, using a REST client such as the Poster add-on for Firefox. You will see that the event handlers are triggered as appropriate. For example:

Figure 2. A GET request on the book resource with an identifier triggers the `onRetrieve()` event handler

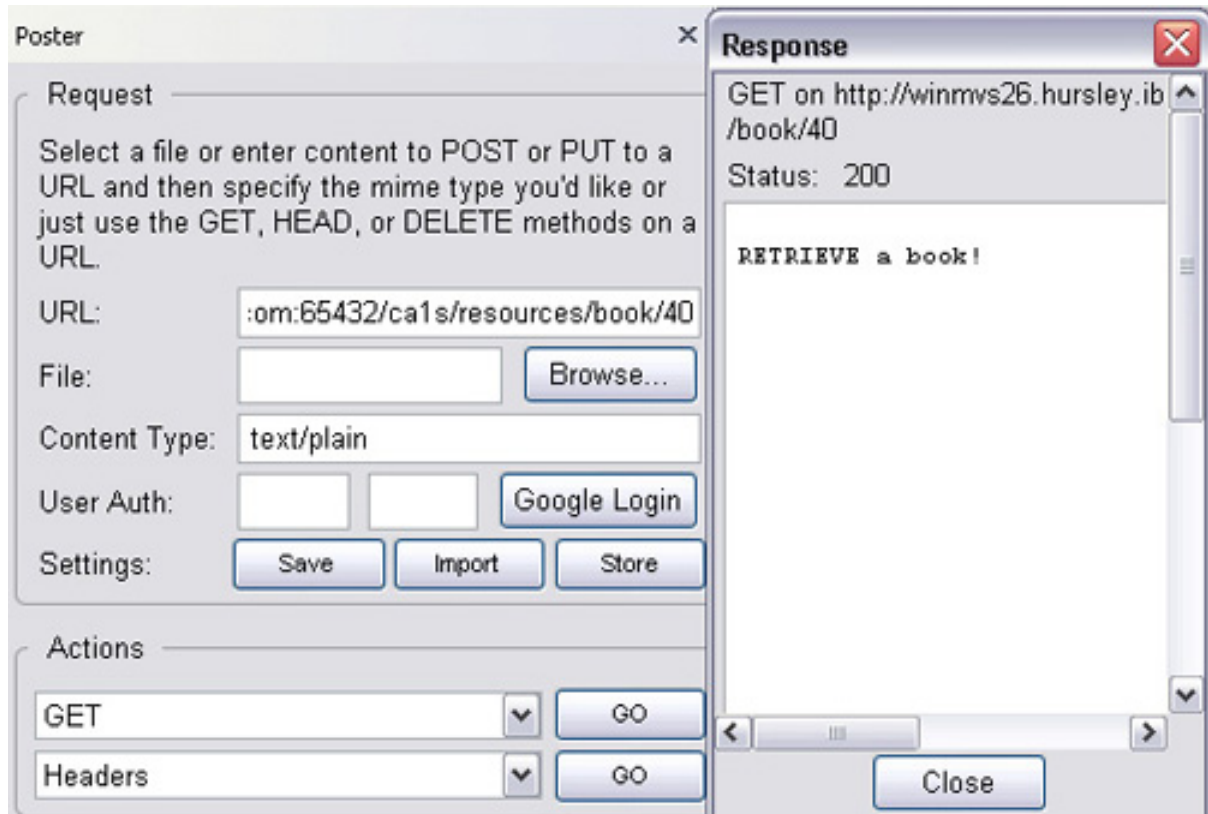
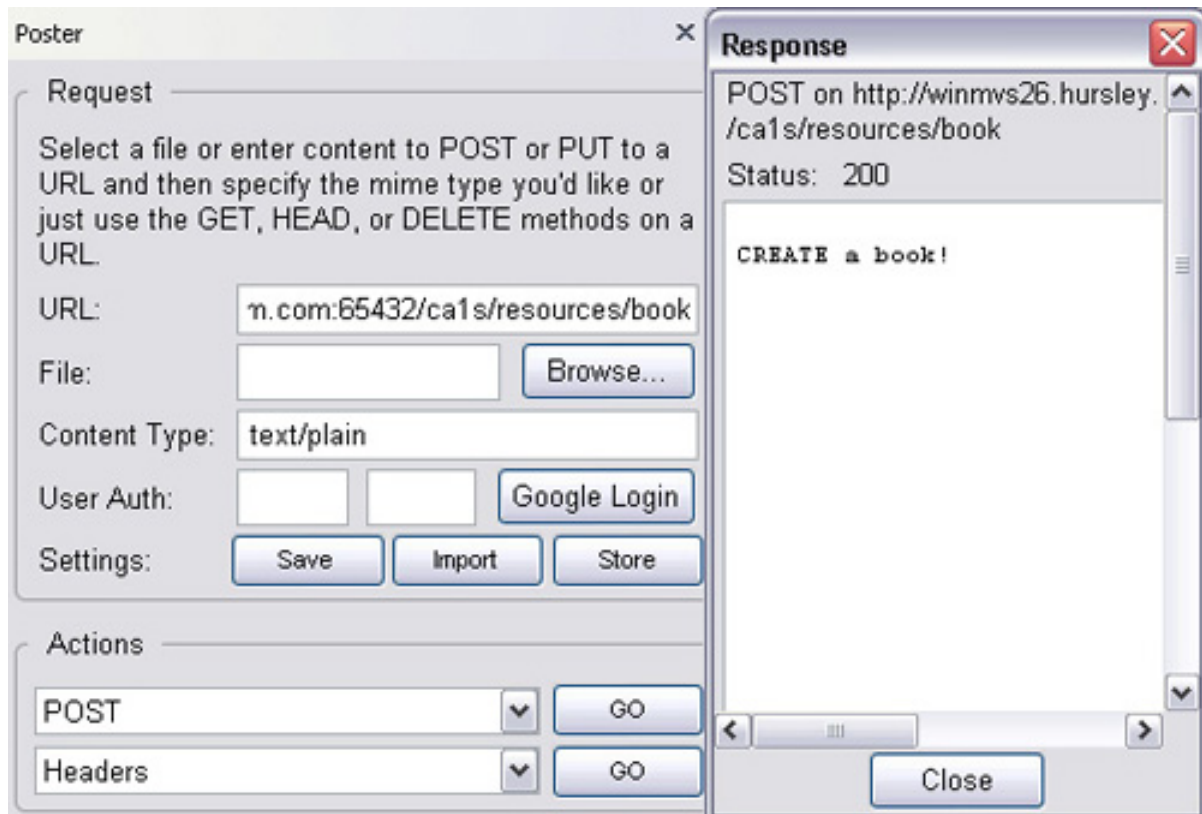


Figure 3. A POST request on the book resource without an identifier triggers the onCreate() event handler



Event handlers not being invoked?

Confirm that the following are all identical (they should all represent the resource name):

- the first path component of the request URI after the 'ca1s/resources/' prefix
- the file name of the resource handler script up to the .php extension
- the name of the class defined in the script

Accessing request data with zget()

Scripts may access various attributes of the inbound request by using the `zget()` function, which is built in to CA1S. Examples include:

- The body of the inbound request as a string, transcoded from the request encoding to the PHP runtime encoding:

```
$data = zget('/request/input/transcoded');
```

- Query string parameters (as well as POST and PUT parameters, if they are form encoded):

```
$value = zget('/request/params/<paramName>');
```

- The ID of the requested resource, as specified on the inbound URI.

```
$id = zget('/request/params/<resourceName>Id');
```

- The part of the URI following the resource ID, if available:

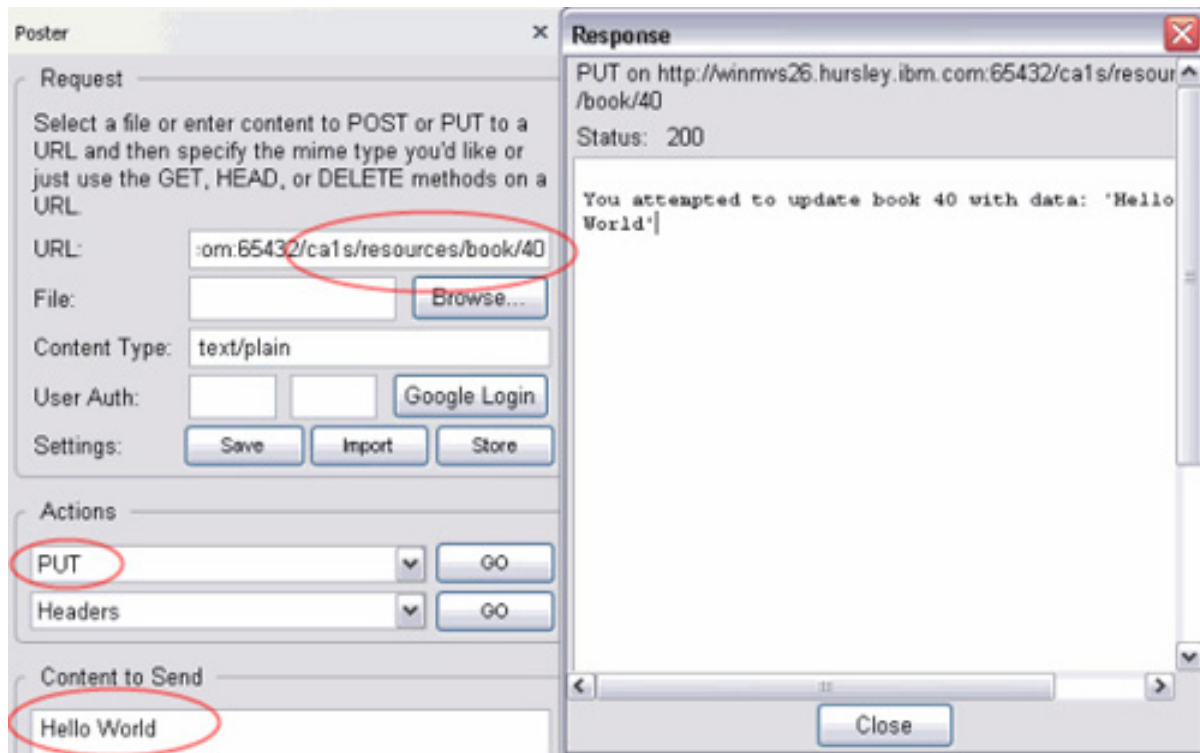
```
$data = zget('/event/pathInfo');
```

To experiment with `zget()`, modify the method `book::onUpdate()` in `book.php` as follows:

```
function onUpdate() {  
    $id = zget('/request/params/bookId');  
    $data = zget('/request/input/transcoded');  
    echo "You attempted to update book $id with data $data";  
}
```

Then, access a specific book resource with a PUT request in order to trigger the update event.

Figure 4. Testing zget with Poster



The `zget()` function is based on the Global Context feature in WebSphere sMash (see [Resources](#)). A description of the full functions provided by `zget()` in CA1S and its differences compared to the version in WebSphere sMash is available in the CA1S documentation. You may also use PHP's standard mechanisms for accessing request data such as the `$_GET` superglobal, the `php://input` stream, and so on.

Using JSON as the service's data interchange format

REST helps define a simple and consistent interface to expose a resource as a service, but it does not impose a format for the service's input and output data—this decision is left to the implementer. JSON is a popular option, because it is simple, human readable, and supported by a wide range of client- and server-side languages.

Encoding PHP variables into JSON data

The following code listing shows how an `onRetrieve()` handler would send a book resource encoded as a JSON string. Note that in this example, for simplicity, the book details are hard-coded in the handler. In the next section, you'll see how to retrieve that information from the COBOL library program.

```
function onRetrieve() {
    $id = zget('/request/params/bookId');
    // ... establish that book with ID $id is Heart of Darkness ...
}
```

```
$book['title'] = 'Heart of Darkness';  
$book['author'] = 'Joseph Conrad';  
echo json_encode($book);  
}
```

The response body returned from a GET request to a member URI like `/cals/resources/book/10` would contain:

```
{"author":"Joseph Conrad","title":"Heart of Darkness"}
```

Decoding JSON data into

Similarly, here's how an `onCreate()` handler would process input data in the request that is encoded as JSON. Again, for simplicity, we simply dump out the decoded data; in a real world case, we would use the data to create a new resource.

```
function onCreate() {  
    $inputData = zget('/request/input/transcoded');  
    $book = json_decode($inputData, true);  
    // ... Create new book resource based on data in $book ...  
    // Dump out the variable to illustrate:  
    var_dump($book);  
}
```

Sending data like

```
{"author":"Margaret Atwood","title":"Oryx and Crake"}
```

in the body of a POST request to the collection URI `cals/resources/book` would return:

```
array(2) {  
    ["author"]=>  
    string(15) "Margaret Atwood"  
    ["title"]=>  
    string(14) "Oryx and Crake"  
}
```

Section 5. Putting it all together

So far you have:

- Established how to interact with CICS COMMAREA programs from PHP
- Learned some of CA1S's conventions and functions that help build RESTful Web services

We will now use this knowledge to expose the library program as a RESTful Web service.

RESTful design

The resource

The first step is to define the resources to which we intend to provide access. In this scenario, we have a single resource type, book, which has four properties:

```
book {
  string: title,    // The title of the book
  string: author,  // The author of the book
  boolean: onLoan, // Whether the book is on loan
  string: borrower // The person who borrowed the book
                  // if it is on loan, null otherwise.
}
```

The methods

Next, let's briefly describe the intent and expected input/output data of each supported method:

Table 6. Requests on the collection URI (/ca1s/resources/book)

HTTP Method	Handler method invoked in book.php	Description
GET	book::onList()	Return the complete list of books and all their properties, encoded in JSON.
POST	book::onCreate()	Add a book to the library. The request must supply the title and author properties of the new book, encoded in JSON. Returns a message indicating that the book was added.
PUT	book::onPutCollection	Not supported
DELETE	book::onDeleteCollection	Not supported

Table 7. Requests on a member URI (/ca1s/resources/book/10)

HTTP method	Handler method invoked in book.php	Description
GET	book::onRetrieve()	Return a single book and all its properties, encoded in JSON. If an individual property is requested (for example, book/10/title), return just that property, encoded in JSON.
POST	book::onPostMember	Not supported
PUT	book::onUpdate()	Mark a book as borrowed or returned. The request must supply a JSON object containing the onLoan and borrower properties. Updating the author or title is not supported. Returns a message indicating that the book was added.
DELETE	book::onDelete()	Permanently remove a book from the library. Returns a message indicating that the book was delete.

Any error cases will result in the service returning the appropriate HTTP status code and an error message in a JSON object property named `errorMessage`. For example, attempting to retrieve a book with ID 909 when there is no such book will return an HTTP 404 response with the following body:

```
{"errorMessage": "Could not retrieve book 909 as it was not found."}
```

Implementation

Finally, let's implement the resource handler `book.php` based on the design above. The full script is included in the sample code package under `resources/book.php`.

Code common to all events

The script `book.php` contains a class `book` with LCRUD handler methods, but it also includes code outside of the class definition. This code will be executed for all requests, before the appropriate handler method is invoked.

In our scenario, it is useful to import the Java classes representing the COMMAREA and the program constants outside of the class definition, because they will most likely be used regardless of the event. We also set the response content-type to `text/json` using the [PHP function `header\(\)`](#), because we know that all requests will return data in JSON format:

Listing 7. Preamble code used by all request types

```
// Load the Java classes
java_import('library.Library_Commarea');
java_import('library.Library_Constants');
// All responses will be JSON, so always set response content-type to text/json.
header('Content-Type: text/json');
```

We also wrap the program link operation and its exception handling in a global function. This function will be available to all event handlers. Should the library program terminate abnormally, the execution of the script will be halted and an error message will be returned to the client.

Listing 8. Wrapper function for program link

```
/**
 * Invoke the library program with the supplied COMMAREA.
 * Notify the client if an error occurs.
 */
function runLibraryProgram($COMMAREA) {
    $program = new CICSProgram('LIBRARY');
    try {
        $program->link($COMMAREA);
    } catch (CICSException $e) {
        header('HTTP/1.1 500 Internal Server Error');
        $error['errorMessage'] =
            'Exception when linking with CICS program:' . $e->getMessage();
        echo json_encode($error);
        exit;
    }
}
```

Finally, the class constructor will also be invoked before any handler, so we can use it to set up properties that are likely to be used by all methods.

Listing 9. The book resource handler constructor

```
/**
```

```

    * The constructor is invoked before event handlers.
    */
function __construct() {
    $this->COMMAREA = new Library_Commarea();
    $this->constants = new Library_Constants();
}

```

Next, let's take a look at the implementation of each of the event handlers.

onList

The `onList()` handler implements similar logic to the `library.php` script that we used to learn about calling COMMAREA programs from PHP: it invokes the library program with a LIST command, then iterates over the collection of books. However, rather than just printing the title and author, this version builds up an associative array representing the books, then serializes it to JSON. It also checks the return code of the library invocation and sends an error message to the client if something went wrong.

Listing 10. The onList handler

```

/**
 * Respond with a JSON-encoded list of books.
 */
function onList() {
    // Attempt to get book list
    $this->COMMAREA->setLibRequestType('LIST');
    runLibraryProgram($this->COMMAREA);

    // Process return code
    switch ($this->COMMAREA->getLibReturnCode()) {
        case $this->constants->getLibraryOk():
            // Return the list of books
            $books = array();
            for ($i = 0; $i < $this->COMMAREA->getLibItemCount(); $i++) {
                $CICSbook = $this->COMMAREA->getLibBookItem($i);
                $bookId = $CICSbook->getBookItemRef();
                $books[$bookId]['title'] = trim($CICSbook->getBookTitle());
                $books[$bookId]['author'] = trim($CICSbook->getBookAuthor());
                $books[$bookId]['onLoan'] = $CICSbook->isBookOnloan();
                $books[$bookId]['borrower'] = $books[$bookId]['onLoan'] ?
                    trim($CICSbook->getBookBorrower()) : null;
            }
            echo json_encode($books);
            break;
        default:
            // Notify client of internal error
            header('HTTP/1.1 500 Internal Server Error');
            $error['errorMessage'] = 'Unexpected return code when listing books: '
                . $this->COMMAREA->getLibReturnCode();
            echo json_encode($error);
    }
}

```

onCreate

The `onCreate()` handler includes an initial step to retrieve and verify the input data

from the request, then follows a similar pattern to `onList()`. The PHP function `header()` (see [Resources](#)) is used to set the appropriate HTTP response code depending on the success of the request.

Listing 11. The `onCreate` handler

```
/**
 * Add book to library based on JSON data in the request body.
 */
function onCreate() {
    // Check input data for new book
    $book = json_decode(zget('/request/input/transcoded'), true);
    if (!is_array($book) || !isset($book['title'], $book['author'])) {
        header('HTTP/1.1 400 Bad Request');
        $error['errorMessage'] = 'Bad book data: ' . zget('/request/input/transcoded')
            . '. Please specify an author and a title.';
        echo json_encode($error);
        return;
    }

    // Attempt to create book
    $this->COMMAREA->setLibRequestType('ADD');
    $this->COMMAREA->getLibBookItem(0)->setBookTitle($book['title']);
    $this->COMMAREA->getLibBookItem(0)->setBookAuthor($book['author']);
    runLibraryProgram($this->COMMAREA);

    // Process return code
    switch ($this->COMMAREA->getLibReturnCode()) {
        case $this->constants->getLibraryOk():
            header('HTTP/1.1 201 Created');
            $bookId = $this->COMMAREA->getLibBookItem(0)->getBookItemRef();
            $status['statusMessage'] = "Successfully created book $bookId.";
            echo json_encode($status);
            break;
        case $this->constants->getLibraryFull():
            header('HTTP/1.1 400 Bad Request');
            $error['errorMessage'] = 'Could not create book : Library is full.';
            echo json_encode($error);
            break;
        default:
            header('HTTP/1.1 500 Internal Server Error');
            $error['errorMessage'] = 'Unexpected return code when creating book '
                . $this->COMMAREA->getLibReturnCode();
            echo json_encode($error);
            break;
    }
}
```

onRetrieve

`onRetrieve()` is similar to `onList()`, but accesses a single book rather than the full list. Furthermore, it includes logic to access an individual property of a book for requests to sub-paths like `book/10/title`. The requested property is determined with `zget('/event/pathInfo')`.

Listing 12. The `onRetrieve` handler

```
/**
 * Respond with a JSON representation of an individual book, or a specific attribute
 * of a book.
 */
```

```

function onRetrieve() {
    $bookId = zget('/request/params/bookId');
    $this->COMMAREA->setLibRequestType('QUERY');
    $this->COMMAREA->getLibBookItem(0)->setBookItemRef($bookId);
    runLibraryProgram($this->COMMAREA);

    // Process return code
    switch ($this->COMMAREA->getLibReturnCode()) {
        case $this->constants->getLibraryOk():
            $book['title'] = trim($this->COMMAREA->getLibBookItem(0)->getBookTitle());
            $book['author'] = trim($this->COMMAREA->getLibBookItem(0)->getBookAuthor());
            $book['onLoan'] = $this->COMMAREA->getLibBookItem(0)->isBookOnloan();
            $book['borrower'] = $book['onLoan'] ?
                trim($this->COMMAREA->getLibBookItem(0)->getBookBorrower()) : null;
            break;
        case $this->constants->getLibraryNotFound():
            header('HTTP/1.1 404 Not Found');
            $error['errorMessage'] = "Could not retrieve book $bookId as it was not found.";
            echo json_encode($error);
            return;
        default:
            header('HTTP/1.1 500 Internal Server Error');
            $error['errorMessage'] = "Unexpected return code deleting book $bookId: "
                . $this->COMMAREA->getLibReturnCode();
            echo json_encode($error);
            return;
    }

    // Send back appropriate info about the book
    $requestedInfo = zget('/event/pathInfo');
    switch($requestedInfo) {
        case null:
            // return the whole book
            echo json_encode($book);
            break;
        case '/title':
        case '/author':
        case '/onLoan':
        case '/borrower':
            // return just the relevant info
            $requestedInfo = substr($requestedInfo, 1);
            echo json_encode(array($requestedInfo => $book[$requestedInfo]));
            break;
        default:
            header('HTTP/1.1 404 Not Found');
            $error['errorMessage'] = "Could not retrieve book detail $requestedInfo
                about book $bookId: don't know what $requestedInfo is.";
            echo json_encode($error);
    }
}

```

onUpdate

The `onUpdate()` handler checks the correctness of the input data, which should contain properties `onLoan` and `borrower`. It then marks books as "borrowed" or "returned" by invoking the CICS program as appropriate. As with previous handlers, the program's return code is checked and error cases are processed as appropriate. Unlike `OnRetrieve()`, this handler doesn't support operations on individual property sub-paths (for example, `book/10/onLoan`); feel free to add this function if you'd like to experiment further!

Listing 13. The `onUpdate` handler

```

/**
 * Update the status of a book to mark it as borrowed or returned.
 */
function onUpdate() {
  // Check input data
  $bookId = zget('/request/params/bookId');
  $updateData = json_decode(zget('/request/input/transcoded'), true);
  if (!isset($updateData['onLoan'])) {
    header('HTTP/1.1 400 Bad Request');
    $error['errorMessage'] = 'Bad book update data: ' .
      zget('/request/input/transcoded') . '. Please specify onLoan.';
    echo json_encode($error);
    return;
  }
  if ($updateData['onLoan'] && empty($updateData['borrower'])) {
    header('HTTP/1.1 400 Bad Request');
    $error['errorMessage'] = 'Bad book update data: ' .
      zget('/request/input/transcoded') . '. Please specify a borrower.';
    echo json_encode($error);
    return;
  }

  // Attempt to update book status
  $this->COMMAREA->setLibRequestType($updateData['onLoan'] ? 'BORROW' : 'RETURN');
  $this->COMMAREA->getLibBookItem(0)->setBookItemRef($bookId);
  $this->COMMAREA->getLibBookItem(0)->setBookLoanStatus($updateData['onLoan']);
  if (isset($updateData['borrower'])) {
    $this->COMMAREA->getLibBookItem(0)->setBookBorrower($updateData['borrower']);
  }
  runLibraryProgram($this->COMMAREA);

  // Process return code
  switch ($this->COMMAREA->getLibReturnCode()) {
    case $this->constants->getLibraryOk():
      $status['statusMessage'] = "Successfully updated status of book $bookId.";
      echo json_encode($status);
      break;
    case $this->constants->getLibraryNotFound():
      header('HTTP/1.1 404 Not Found');
      $error['errorMessage'] = "Could not update book $bookId as it was not found.";
      echo json_encode($error);
      break;
    default:
      header('HTTP/1.1 500 Internal Server Error');
      $error['errorMessage'] =
        "Unexpected return code when updating status of book $bookId: " .
        $this->COMMAREA->getLibReturnCode();
      echo json_encode($error);
  }
}
}

```

onDelete

Lastly, the `onDelete()` handler invokes the CICS program to perform a 'DELETE' operation on the book ID supplied in the request URI.

Listing 14. The onDelete handler

```

/**
 * Delete a book from the library.
 */
function onDelete() {

```

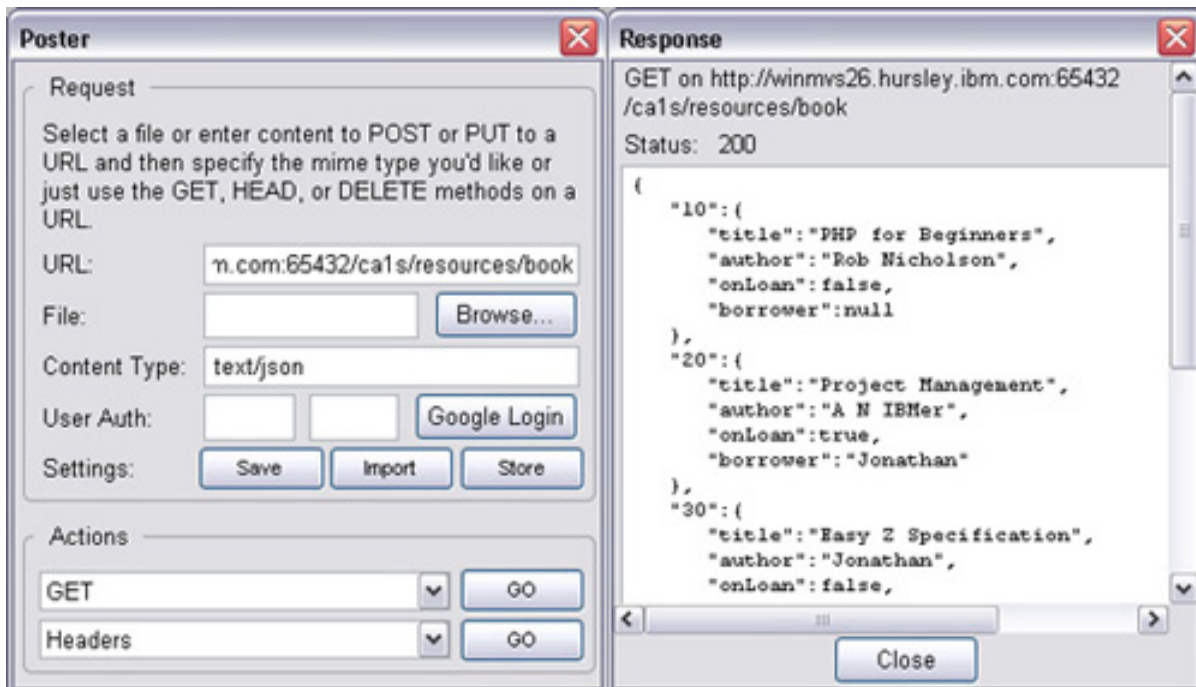
```
// Attempt to Delete book
$bookId = zget('/request/params/bookId');
$this->COMMAREA->setLibRequestType('DELETE');
$this->COMMAREA->getLibBookItem(0)->setBookItemRef($bookId);
runLibraryProgram($this->COMMAREA);

// Process return code
switch ($this->COMMAREA->getLibReturnCode()) {
    case $this->constants->getLibraryOk():
        $status['statusMessage'] = "Successfully deleted book $bookId.";
        echo json_encode($status);
        break;
    case $this->constants->getLibraryNotFound():
        header('HTTP/1.1 404 Not Found');
        $error['errorMessage'] = "Could not delete book $bookId as it was not found.";
        echo json_encode($error);
        break;
    default:
        header('HTTP/1.1 500 Internal Server Error');
        $error['errorMessage'] = "Unexpected return code deleting book $bookId: "
            . $this->COMMAREA->getLibReturnCode();
        echo json_encode($error);
}
}
```

Consuming the Web service

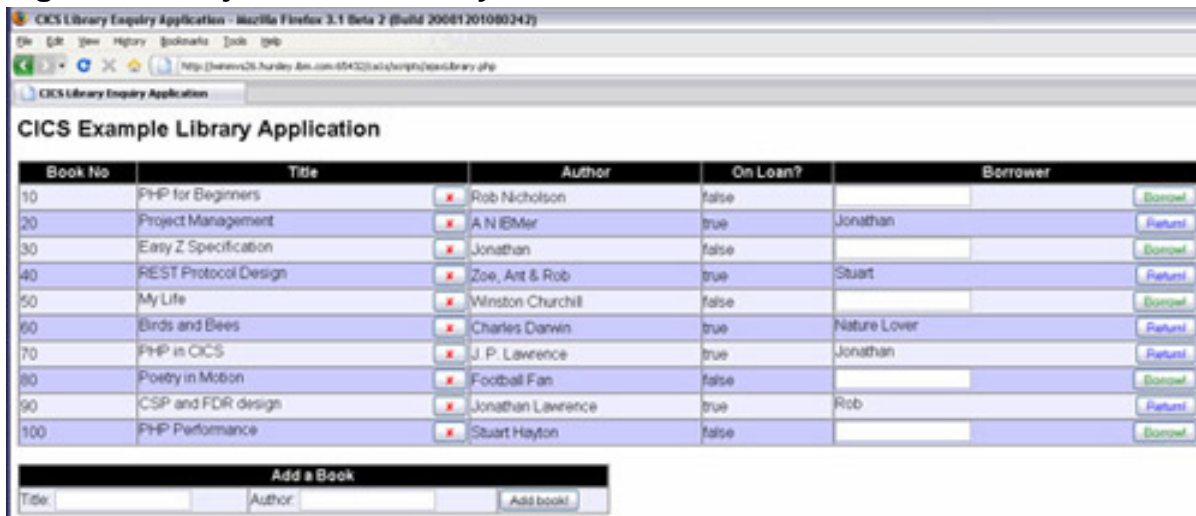
Now that the library application is exposed through a consistent interface and communicates using clearly defined JSON data structures, it can easily be accessed by a wide range of clients. The simplest way to test the service is with a lightweight REST client like the Poster add-on for Firefox (see [Resources](#)).

Figure 5. Testing your service with a simple REST client



In the sample code, we include an example HTML and Javascript page that issues Ajax requests to the service. You can try it out by copying library/scripts/ajaxLibrary.php to your CICS system under /ca1s/work/scripts/ajaxLibrary.php (and, if you haven't done so already, transfer resources/book.php to /ca1s/work/resources/book.php) then access it in your browser.

Figure 6. An Ajax front-end for your service



The library service could also be mashed-up with other services using a platform like WebSphere sMash.

Section 6. Summary and next steps

In this tutorial, you have learned:

- How to generate Java classes representing the CICS COMMAREAs and make them available to CA1S.
- How to prepare input data for a CICS COMMAREA program, invoke the program, and read its output data using PHP code.
- How to use PHP reflection to explore the fields available on a COMMAREA object.
- How to use WebSphere sMash-style event handlers and `zget()` to quickly write PHP code that responds to REST requests.
- How to encode and decode JSON data from PHP.
- How to use the knowledge to expose a CICS program as a RESTful Web service.

Next, you could explore the features of CA1S that were not covered by this tutorial, such as:

- Accessing DB2 databases with the PDO extension
- Managing units of work and transactionality from PHP code using the Commit/Rollback API
- Debugging PHP scripts with PDT

For more information, see the CA1S documentation.

Downloads

Description	Name	Size	Download method
Sample zip file	CA1S.devworks.sampleCode.zip	9KB	HTTP

[Information about download methods](#)

Resources

Learn

- Get more [information on SupportPac CA1S](#), or [download it](#) and access the user guide.
- Visit the [CA1S forums](#) to get answers and give feedback on the SupportPac.
- Watch a [video of SupportPac CA1S in action](#).
- See the [documentation for CA1S](#).
- Learn more about CICS Transaction Server at the [CICS TS 3.2 Information Center](#).
- Find out about the [IBM JZOS Batch Toolkit for z/OS SDKs site at alphaWorks](#).
- See documentation on the [PHP reflection classes](#).
- See documentation for the the [PHP function `get_class_methods\(\)`](#).
- See documentation for the the [PHP function `phpinfo\(\)`](#).
- Learn more about PHP with the [PHP language reference](#).
- Read the [WebSphere sMash documentation](#).
- Download the [CA1S documentation](#).
- Read about the [Global context feature in WebSphere sMash](#).
- Check out the home for [JavaScript Object Notation \(JSON\)](#).
- See documentation on the The [PHP function `header\(\)`](#).
- Check out the [developerWorks PHP zone](#) for comprehensive PHP project resources.
- For more information on WebSphere sMash, which shares the PHP scripting technology and REST related conventions used in CA1S, see [ProjectZero](#), the community site where you'll find blogs, forums, and a community of fellow developers.
- For more information on the Java Bridge, which allows PHP scripts to interact directly with Java classes on CA1S and WebSphere sMash, see this [developerWorks article](#).

Get products and technologies

- Download the [Eclipse PHP Development Tool \(PDT\)](#).
- Download the [Target Management plug-in for Eclipse](#).

- Test your RESTful Web services, with the REST client with [Poster add-on for Firefox](#).

About the authors

Robin Fernandes

Robin Fernandes joined IBM's Java Technology Centre in Hursley, United Kingdom as Software Developer after graduating from Imperial College in 2003. His current focus is a Java-based runtime for PHP, which is used in the CA1S SupportPac and WebSphere sMash. He also regularly contributes test cases and patches to php.net and enjoys experimenting with audio software in his spare time.

Jonathan Lawrence

Jonathan Lawrence joined IBM's Java Technology Centre in Hursley, United Kingdom as a Software Developer in 2006, after 4 years in Hursley's Software Services department where he was a CICS and Cross-platform Integration Specialist. He designed the CICS integration aspects of the CA1S SupportPac.