CICS update

241

December 2005

In this issue

3 CICS batch application control
13 Establishing a Web services infrastructure within CICS
30 BigCommarea utility user guide – part 2
44 CICS questions and answers
46 CICS news

A tcl PUBLICATION

CICS Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *CICS Update*, comprising twelve monthly issues, costs $270.00 in the USA and Canada; £175.00 in the UK; £181.00 in Europe; £187.00 in Australasia and Japan; and £185.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the December 2001 issue, are available separately to subscribers for $24.00 (£16.00) each including postage.

CICS Update on-line

Code from *CICS Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon.com/cics; you will need to supply a word from the printed issue.

Contributions

When Xephon is given copyright, articles published in *CICS Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

CICS batch application control

In line with its current trend of strengthening CICS with an elaborate tools portfolio, IBM has released its latest tool – CICS Batch Application Control, also referred to as CICS BAC.

Gone are the days when the CICS regions are brought down before starting the batch run. With growing demand for increased availability of online working, it becomes essential for batch and CICS to co-exist and share resources. IBM CICS BAC is the right tool to enable the easy management of this resource sharing between CICS and batch.

CONCEPT OF BAC

CICS provides dynamic control of CICS file control, transient data, transaction, and program resources. BAC uses these CICS control requests to allocate, release, and re-allocate resources associated with CICS applications.

It's usually the CICS administrator, with BAC (or an equivalent tool), who takes resources offline in CICS so that the batch process can access them.

The core of BAC is the batch request utility, which runs as a standard batch job step and enables you to control the CICS resources within multiple CICS regions. These batch jobs can:

- Issue control requests that process either individual resources or groups of resources.
- Invoke processes that use the resources while they are released.
- Invoke CICS programs and transactions, including CEMT.

BAC COMPONENTS

The BAC architecture consists of six major components: BAC control file, BAC workstation administration client, BAC communications server, BAC start-up processor, BAC request server, and BAC batch request utility.

BAC control file

Each CICS region has a single BAC control file – a VSAM KSDS – in which all the control and state information is maintained. When the CICS region changes the state of a resource being tracked by BAC (using CEMT or **EXEC CICS SET** commands), the control file is updated using the BAC Global User Exit.

Similarly, when the BAC batch request utility requests a CICS region that is not active, the batch job step itself allocates, opens, and updates the state information in the CICS region's control file.

In effect, the control file information is updated at the time a request is made by the CICS region, the BAC communication server, or the batch request utility job step. This ensures that the correct status of the resources is always maintained.

Access to the control file is serialized by a system-level enqueue obtained by the CICS BAC start-up processor. This ensures that while the BAC control file is open in CICS, it cannot be opened by the batch utility.

It is interesting to note that BAC allows you to control resources not explicitly defined in the CICS region control file (ie the CICS BAC batch request utility program can be used to close and disable a file that doesn't have an entry in the CICS region control file).

BAC workstation administration client

CICS BAC provides a Windows-based workstation administration client to maintain all the options and definitions in the control file.

The client can be used to create and modify objects including region properties, file, TD queue, program, transaction, group, and list. Once the objects are created, they should be uploaded to the control file on the host. Similarly, when modifying an object, it has to be downloaded from the host, modified, and then uploaded.

In addition, the client can be used to browse or delete objects from the control file. The client can also be used to perform CICS INQUIRE functions.

The client communicates with the CICS BAC communication server – identified by a unique server identifier – through a TCP/IP connection.

BAC communications server

The CICS BAC communications server channels requests from the workstation client to the appropriate CICS region to update its CICS BAC control file. If the CICS region is not available, the communication server itself attempts to allocate and open the control file and process the request, provided the request does not require direct access to CICS resources.

The CICS BAC communication server, which runs in its own MVS address space, can be initiated either as a start-up task or as a batch job. There is no direct link between a communication server and a specific CICS region – in the sense that a communication server can communicate with different CICS regions.

Similarly, multiple communication servers can run on a single MVS image and these different servers can communicate with the same CICS region.

BAC start-up processor

The CICS BAC start-up procedure – typically initiated by an entry in the CICS region PLTPI – ensures that all resources reflect their states as known to CICS BAC by means of the control file.

For example, a batch request utility can set a file status to ‘closed’ in the BAC control file, when a CICS region is not up. If the CICS region is brought up before the batch request utility resets the file status to ‘open’, the BAC start-up procedure ensures that the file remains closed in CICS.

The start-up processor also starts the CICS BAC request server subtask. When the start-up processor has completed its work, it terminates.

BAC request server

CICS BAC request server, which runs as a standard CICS application program, is initiated by a CICS Distributed Program Link (DPL) request using the external CICS interface (EXCI). BAC request server handles all the requests – originated in a BAC communications server or a batch job step running the BAC batch request utility – sent to a CICS region.

A BAC request server subtask gets started by the BAC start-up procedure and then runs in the CICS address space as long as BAC is active in the CICS region. The updates to the control file are performed by this subtask, ensuring that the control file is allocated uniquely to the CICS region and preventing direct access to the file by the communication server or a batch request utility.

BAC batch request utility

The CICS BAC batch request utility runs as a job step in a batch job stream and can be used to issue commands to one or more CICS regions to ensure that the state of a given resource is as requested.

The batch request utility communicates with the CICS request server using DPL requests over EXCI links. If CICS BAC is not active in the target CICS region, the batch request utility updates the appropriate records in the CICS region control file to reflect the required resource state.

HOW DOES BAC WORK?

In a regular batch job, if you need to access the data that is typically open and online in a CICS region, you can include the BAC batch request utility program as a job step in your JCL. Using the BAC batch utility commands you can close the dataset that is open in CICS.

For example, the sequence of steps could be as follows:

1 CICS BAC job step passes a **CICS BAC SET FILE(filename) OPENSTATUS(CLOSED) ENABLESTATUS(DISABLED)** command as an EXCI DPL request to the CICS target region.

  The region can either be specified explicitly in the command or it can be the default region set in an earlier **CICS BAC DEFAULT** command.

2 CICS receives the DPL request and executes the CICS BAC request server program to process the command, which results in an **EXEC CICS SET FILE CLOSED DISABLED** command.

  The disabled option requires that the file must be explicitly enabled before it can be used again by CICS. This ensures that the file cannot be opened implicitly by a CICS transaction, say by issuing an **EXEC CICS READ** command.

3 Access to the CICS BAC control file is handled by the CICS BAC request server subtask (CBKCRHST), which updates the status of the file record in the control file to CLOSED DISABLED.

4 The CICS BAC request server returns the results of the command to the CICS BAC batch request utility.

5 If the command has executed successfully, the batch job can continue with the main job step to process the file.

6 Once the batch job successfully completes its function, the file can be re-opened for CICS again using a CICS BAC job step.

SETTING UP BAC

The major steps involved in setting up BAC are as follows.

Create the BAC control file

Create a unique CICS BAC control file for each of the CICS regions requiring BAC support. The KSDS can be defined and initialized using the sample job CBKDIFIL in the SCBKSAMP dataset.

Setting up CICS BAC components

Add BAC load module to CICS

When installing CICS BAC, the load modules are installed in the SCBKLOAD library. Make these load modules available to CICS by adding SCBKLOAD to the DFHRPL concatenation in the CICS region JCL (note that this requires the region to be restarted before the additions can be used).

Define BAC resources to CICS

BAC resources can be defined by using either CICS CEDA or the DFHCSDUP batch utility program and the sample JCL available in the SCBKSAMP dataset:

1 Define the CICS BAC transaction KBKM so that you can activate and control CICS BAC in the CICS region. Note that while it is OK to change the first three characters 'KBK', in case it conflicts with the installation standards, it is necessary that the last character be retained as 'M'. CBKTRANS is the sample JCL in the SCBKSAMP dataset that can be used with the batch utility for defining them.

2 Define the CICS BAC programs and mapsets to CICS, again using CEDA or the batch utility. CBKPROGS is the sample JCL in the SCBKSAMP dataset that can be customized for this.

3 Define the CICS BAC control file (CBKCNTL) to CICS

using CEDA or a batch utility after customizing the CBKFILES in the sample dataset.

Install BAC resources in CICS

After defining the BAC resources, install them using the **CEDA INSTALL GROUP(cbkgroup)** command, where *cbkgroup* is the group name specified in the resource definitions.

For BAC resources to be automatically installed during CICS initiation, you can add the *cbkgroup* to a CICS start-up list – again using the CEDA command.

Start BAC in the CICS region

To start CICS BAC during CICS region initialization, add an entry for CBKCMNDS (CICS BAC command processor program) to the PLTPI table. Note that this entry is immediately after the DFHDELIM entry, to ensure that it is executed before any other program that could access the resources under BAC.

Alternatively you can use the **START** command along with the CICS BAC transaction KBKM to start the CICS BAC component in the CICS region.

Shutting down BAC in the CICS region

To shut down CICS BAC automatically during CICS shutdown processing, add an entry for CBKCMNDS to the PLT shutdown table. Ensure that it is the last entry in the PLT.

Alternatively you can use the **SHUTDOWN** command along with the CICS BAC transaction KBKM.

CICS region commands for CICS BAC

To control the CICS BAC component that runs in a CICS region, the CICS BAC commands **START**, **SHUTDOWN**, **DUMP**, and **STATUS** can be used.

The general format for a CICS BAC request server command is:

```
KBKM COMMAND, parm(value)
```

where *KBKM* is the CICS BAC command processor transaction ID, *COMMAND* is one of the CICS BAC component commands, and *parm(value)* is an optional parameter specified on the command.

Define CICS BAC datasets

Even when CICS BAC is not active, the BAC communication server and BAC batch request utility can access the BAC files. Hence the following datasets have to be defined:

1 Control file table – the control file table member name must be CBKCFTBL and the fully-qualified dataset name of the control file of each CICS region has to be defined.

2 Run-time parameters – CBKPARMS member, optional, depending on whether you need to provide the parameters.

3 Audit log dataset – the CICS region and the batch request utility both use the audit log, DDname CBKLOG, to record the results of processing batch request utility requests.

Setting up and starting the BAC communication server

The list of the tasks necessary to set up a CICS BAC communication server are:

1 Customize the sample communication server started task procedure using the sample JCL CBKCSRVR in the CBKSAMP library.

2 If the communication server has to perform security checking for workstation requests, ensure that SCBKLOAD and SDFHEXCI datasets are APF-authorized or are included in the linklist concatenation.

3 Ensure that the control file table CBKCFTBL member is available in either the CBKPARMS dataset or a dataset in the MVS logical parmlib concatenation.

4 Specify the run-time parameters if you don't want the default values.

5 Ensure the parameters that the communication server needs are provided in the CBKSRVR member in the CBKPARMS dataset, or another parmlib dataset in the MVS logical parmlib concatenation.

6 Ensure that the communication server ID (that is, the user ID that the server runs under) is authorized to access TCP/IP services.

7 Start the communication server started task or job.

Setting up and running the BAC batch request utility

While a batch job requires access to resources allocated to CICS, the CICS BAC batch request utility program can be invoked as a job step.

CICS BAC batch request utility job steps require the following statements:

1 An EXEC statement for program CBKBMAIN.

2 A DD statement for the CBKPARMS dataset containing:

- the control file table member, CBKCFTBL, identifying the control file dataset name for each CICS region.
- the parameter member, CBKBATCH, containing the batch request utility run-time parameters.

3 A STEPLIB DD statement that references the SCBKLOAD and the SDFHEXCI datasets containing the load modules required by the batch request utility.

4 A DD statement for the CBKIN dataset, from which the CICS BAC batch request utility reads the commands it is to execute.

5 A DD statement for the CBKPRINT dataset, to which the CICS BAC batch request utility writes its output messages.

Sample JCL for the batch request utility job step can be found in CBKSAMP1 of the SCBKSAMP library.

BAC BATCH REQUEST UTILITY COMMANDS

The basic format of a CICS BAC batch request utility input command is as follows:

```
COMMAND NAME Parm1(value1),Parm2(value2),...,ParmN(valueN)
```

Batch request utility commands:

- DEFAULT – to set the default values for parameters to be applied to subsequent input commands, for example the target CICS region.

  You can specify multiple DEFAULT commands in the same input file. Each DEFAULT command completely overrides the values specified on any previous DEFAULT command.

- LINK – to execute a program in the target CICS region.

  The LINK command is passed to a CICS BAC request server, which in turn links to the requested program. So, the linked-to program runs as part of the same EXCI mirror transaction that is running the CICS BAC request server.

- RUNCEMT – to issue a CEMT command in the target CICS region.

  The CICS BAC batch request utility writes up to 256 bytes of the data returned by the CICS region from the CEMT command to the print dataset, CBKPRINT.

- SET FILE – to set the file options and process a file.

  Either the OPENSTATUS or the ENABLESTATUS parameter must be specified.

  The SET FILE command enables you to open, close, enable, or disable a VSAM file in a CICS region.

- SET GROUP – enables you to set the attributes of all the objects, files, TD queues, transactions, and programs, in the specified group.

- SET LIST – the SET LIST command operates on a list of groups, all of which are processed as in the SET GROUP command.
- SET PROGRAM – to set the executable status of a program.
- SET TDQUEUE – to process a transient data (TD) queue.
- SET TRANSID – to set the executable status of a transaction.
- START – to start a transaction.

CONCLUSION

CICS BAC is yet another tool from IBM that CICS administrators and operations personnel have needed for quite some time. By using this tool, the sharing of resources between CICS and batch can be automatically and effectively managed, and it maximizes the availability of resources.

Sasirekha Cota
System Software Group
Tata Consultancy Services (India)

Establishing a Web services infrastructure within CICS

CICS Transaction Server for z/OS, Version 3 Release 1, provides a fully-integrated environment that supports workloads architected around the Web services programming model. This article outlines the infrastructure used to support a Web service-based workload under the control of CICS. It describes the various resources that need to be configured and provides examples, which can be used to set up and then to test a basic configuration. This may then be used as the foundations on which a more complex workload can be deployed.

The examples shown throughout this paper have been taken from ones used to test the Web Services Atomic Transaction (WSAT) function when it was being developed for CICS Transaction Server 3.1. The business logic has been kept to a minimum, allowing the emphasis to be placed on building a working infrastructure rather than illustrating the power of Web services themselves. WSAT is not discussed in this article, and a follow-on article will be used to explain how this configuration can be extended to support components that form part of Atomic Transactions.

These examples illustrate how CICS can take on the roles of a Web service requester and a Web service provider. It is unlikely that CICS would be used in this manner because there are other, more efficient, ways for business logic, running under the control of CICS, to interact together. However, when CICS undertakes both roles, it is possible to test the infrastructure without the need for any other products.

WEB SERVICES OVERVIEW

Web services technology provides support for the deployment and access of business function components via the World Wide Web. It allows programs to be combined quickly and easily to form more complex functions.

This is achieved by the provision of a common program-to-program communication model that is built on a mixture of existing and emerging standards such as HTTP, SOAP, XML, and WSDL.

WEB SERVICES SUPPORT WITHIN CICS

Business logic can be encapsulated by the Web services architecture to hide its internal implementation from the programs that invoke it across the World Wide Web. CICS business applications typically receive input and generate output by using COMMAREAs (which have been available for many releases of the product) or channels (which are another new feature of CICS Transaction Server 3.1). Internally, a CICS application might use one or more data structures to describe the input and output information.

For a CICS application to be enabled as a Web service its input and output characteristics, which are defined in the data structures it uses, have to be described using the Web Services Definition Language (WSDL). CICS provides a Web services assistant that allows WSDL to be generated from application data structures written in COBOL, PL/I, or C/C++. Some application data structure components cannot be mapped directly into WSDL by the assistant, and, as a result, some applications need to be linked-to via a wrapper program that completes the data mapping process. This article refers to a data structure that can be successfully converted by the Web services assistant.

Any CICS application that wishes to invoke a Web service must be able to pass input data to that service in a form that it understands. The Web services assistant provides a means of generating a data structure from the WSDL describing the Web service, which can then be used by applications wishing to invoke this Web service. Once again, the limitations in the mapping between WSDL and language data structures mean that in some cases these CICS applications will need to link to a wrapper program to complete the mapping process. The application referred to in the example in this article does not, however, need a wrapper program.

When an application invokes a Web service a message is built, which may contain some input data, and this is sent though the network to the provider of the Web service. CICS supports two ways of doing this. One permits the maximum amount of control to be given to the business application, which directly builds the message and then links to the CICS supplied program DFHPIRT from where the message is dispatched into the network. Alternatively, CICS provides a new Application Programming Interface (API) to achieve this. The **INVOKE WEBSERVICE** command allows a business application to use CICS to generate a SOAP message and then send it to the Web service provider. If the business logic is not to be changed, and this command is to be used to create a Web service request, then the business logic must contain an **EXEC CICS LINK** command to allow it to pass control to a wrapper program that then invokes the Web service. The example shown in this article uses business logic that includes the **INVOKE WEBSERVICE** command.

The CICS resources used for Web services support

Several new resource types are introduced in CICS Transaction Server 3.1. Three of these are used to support access to Web services: PIPELINEs, WEBSERVICEs, and URIMAPs.

A PIPELINE is a resource describing a set of message handler programs to CICS, which are driven to build and send a Web service request as a SOAP message. Some of these programs process the message headers, while others process the message body. When a message response is returned to the pipeline, these message handlers are driven again in the reverse order.

Each pipeline is customized using an XML configuration file located within the local Hierarchical File System (HFS). The fully-qualified name of this file is specified as an attribute of the PIPELINE resource definition. This file can be modified to allow additional message handler programs, provided by the user, to be included in the pipeline. No programs are needed by the pipelines in addition to those used to run the examples shown in this article.

The PIPELINE resource also includes the names of two additional HFS directories. The first is a shelf directory, which can be used to introduce support for new Web services to a pipeline while it remains in use. This directory is not needed to run the examples in this article. Secondly, there is a wsdir directory, which is used to store those binding files that a pipeline manager needs to process either inbound or outbound Web service requests. The binding files referred to in the examples in this article are those created using the Web service assistant programs.

The WEBSERVICE resource contains information about a deployed Web service. When a PIPELINE resource is installed, its wsdir directory is scanned and, if CICS finds any Web services binding files there, it creates and installs the appropriate WEBSERVICE resources from information found in these files.

Customers may choose to create their own WEBSERVICE resource definitions and, if they do so, the information in them must match that in the corresponding binding files if the Web service applications they refer to are to be invoked correctly. The example in this article makes use of those WEBSERVICE resources that are generated when a PIPELINE is installed.

The third resource type is the URIMAP, which is used to map a portion of the URL associated with a message that arrives at a PIPELINE with a specific Web service.

Application

The Web service assistant comes as two separate programs – DFHLS2WS and DFHWS2LS. The first of these can be run against language structures that describe the input and output data relating to a program that is being deployed as a Web service. The components that it produces are a WSDL file describing the input and output interfaces of the Web service and a binding file that CICS uses to access the Web service.

The second program can be run against a file containing a WSDL description of a Web service. This produces an equivalent language structure that can be incorporated into those applications that have to provide data to a Web service that they then call, or which needs to understand data that is returned following the running of a Web service. A binding file is also produced, which the CICS pipeline requires to allow it to build a Web service request message, and again to turn a Web service response message into a form that the application understands.

WEB SERVICE REQUESTERS AND PROVIDERS

As we have seen, CICS can take on one of two roles with respect to Web services. When it runs a workload that wishes to make use of remote Web services, CICS acts as a requester. When it supports business logic that has been deployed as one or more Web services, CICS takes on the role of a provider.

CICS acting as a provider of Web services

When CICS is acting as a provider, it makes business logic available via a suitably-configured pipeline. Requests arriving through the network at such a pipeline are examined by the pipeline manager to see whether they map to an installed WEBSERVICE. Any message headers are processed by the pipeline's header handler programs, and the message body can be processed by message handler programs. If this processing is successful then information in the binding file associated with the WEBSERVICE is used to extract the data from the message body and build it into a COMMAREA.

The business logic is then linked to, passing it this COMMAREA. The business logic runs and may return something in a COMMAREA, or produce an abend.

If an abend occurs, the pipeline manager builds a SOAP fault message. If instead data is returned from the business logic in a COMMAREA, the pipeline makes use of information found in the same binding file and uses it to build a message body including the COMMAREA data within it. There is a third possibility where no error occurred while the business logic executed and no data is to be returned from it. Under these circumstances an empty message body is created.

Any header handler or message handler programs that were called when the inbound message was received are then called again in the reverse order, to add or modify message headers, before it is dispatched through the network to the invoker of the Web service.

The CICS Web services provider components

A small number of components are needed to allow CICS to act as a Web services provider.

The CICS resource group DFHSTAND, found in DFHLIST, must be installed. Three other CICS resources are also needed: a PROGRAM definition for the business logic application, a PIPELINE, and a TCPIPSERVICE to allow requests to be received from the network and delivered to the business logic.

In addition to this, three files need to be placed in the Unix file system – the PIPELINE configuration file, the binding file, and a WSDL file describing the business logic as a Web service.

Lastly, Web service messages are transferred though the network in Unicode, as UTF-8 data. They are processed by CICS as Latin1 EBCDIC. Character conversion between these encodings is controlled by CICS and makes use of z/OS Character Conversion Services. This z/OS service must be enabled for CICS to use and needs to be configured to support the conversion of character data between Character Coding Schema IDs (CCSID) of 1208 and 037, and again between 037 and 1208, to allow data conversion to take place in both directions. The *z/OS Support for Unicode* manual, SA22-7649, explains how to carry out these operations.

Configuring CICS as a Web services provider

To complete the steps described in this section, you will need a CICS Transaction Server V3.1 region, a segment in its local

Hierarchical File System (HFS) where various files and directories can be added, and an installed version of the Java run-time environment in the form of the Software Development Kit (SDK) 1.4.2 on z/OS. Once these are available you can proceed with the following instructions.

Example 1, found at the end of this article, is a simple piece of business function that can be deployed as a Web service. This COBOL program uses a single COMMAREA for both its input and output data. The structure of the COMMAREA is a four-character field that is used to pass in a value representing an integer. The program then adds 1 to this number and returns the result as a four-character value. No overflow conditions are handled, so the input data has to be in the range 0000 to 9998.

Copy the program source code to a PDS member. The one I created was called ADDONE00. Compile and link this member into a library where it can be found by your CICS region. If program autoinstall is not active, create and install a PROGRAM resource definition for it as well.

This program has a language structure in its linkage section that describes the COMMAREA. This structure has been copied as Example 2, and the first level has been removed so that it corresponds to the form accepted by the DFHLS2WS application program. When run against this COMMAREA, the application creates a WSDL file and binding file that CICS needs to make Example 1 available as a Web service.

Before running this application, allocate directories in the local HFS where these files and other components can be created. Those that were used for my testing had the following names:

- */u/myid/cicsts/webservices/config/*
- */u/myid/cicsts/webservices/shelf/provider1/*
- */u/myfile/cicsts/webservices/wsdir/provider1/.*

Copy Example2 into a member of a PDS. The one I created was named ADD1COMM.

Next locate the sample JCL that CICS provides to allow the DFHLS2WS application program to be run. This is supplied in the SDFHINST library, and can be copied to a working directory where it can be used to run the application.

Take the JCL, shown in Example 4 and modify it so that the application program can be located from the JCLLIB statement. A number of symbolic parameters may need to be altered to resolve various library members in the local HFS. In Example 4 TMPFILE declares the name of a file in the */tmp* directory, JAVADIR is added to */usr/lpp/* to locate the installed copy of Java, PATHPREF locates the CICS route directory, and USSDIR, which extends this directory, has been intentionally left blank.

The PDSLIB statement in Example 4 needs to identify the location into which you have copied Example 2. The REQMEM and RESPMEM are the names of two structures that the application produces and places in the PDSLIB. Names have been provided in Example 4 because they are compulsory. However, they are not used by the other components of this test.

Run this JCL and, assuming it executes successfully, look in directory number 3 to see that the binding file and WSDL files have been stored there.

A PIPELINE resource needs to be created to allow access to the Web service. The one I used was called PIPEA1P and was defined with STATUS set to ENABLED, and CONFIGFILE, SHELF, and WSDIR naming directories 1, 2, and 3, shown earlier in this article.

A configuration file is needed with this pipeline. Example 6 is the one used for my testing. This must be placed in the directory named by the CONFIGFILE property of the PIPELINE resource definition.

Install this pipeline and look in MSGUSER for some messages that indicate the Web service is installed and ready for use. Message DFHPI0703 shows the start of the install processing taking place. This should be followed by DFGPI0901, showing the WEBSERVICE ADDONE00 being created, followed by a DFHPI0910 indicating that this succeeded. A DFHPI0915 should appear next, showing the WEBSERVICE is in service and ready for use. Then comes a DFHPI0903 message that shows the URI map being created. Finally a DFHPI0704 message should indicate that a scan of the WSDIR directory has taken place and in doing so, one bind file was found, one Web service was created, and no failures occurred.

One final resource is needed to make the new Web service available for testing. A TCPIPSERVICE needs to be added to define the port number and protocol used to deliver Web service request messages into CICS. The one I used specified port number 9185 although there is nothing significant about this number, and set the protocol to HTTP, the URM to DFHWBADX, and the transaction to CWXN. Default values were taken for all the remaining attributes.

CICS behaving as the requester of Web services

As we have seen, there are two ways in which an application, running under the control of CICS, can invoke a Web service. This section describes the processing that takes place when a CICS pipeline is used to automate the building and processing of Web service request and response messages.

A remote Web service must be defined to CICS before it can be invoked by an application running in the local region. Each Web service is made available using a PIPELINE resource, together with a binding file, and a language data structure.

The binding file can be generated from the remote Web service's WSDL. The Web Services Assistant provides a means of achieving this. In addition, the Assistant can be used to generate language data structures from the WSDL, which application programs may then use if they have to provide input to or receive output from that Web service.

Requester pipelines permit user applications to communicate with remote Web services. Each pipeline resource can support more than one Web service. The pipeline manager makes use of a number of data containers within the channel associated with a particular request that it is processing. Information in these containers is used to construct the SOAP message that then invokes a particular Web service. There is a DFHWS-DATA container within this channel that is used to pass input data to the Web service from the calling application, and is where any output data is placed once the Web service has responded.

An application makes use of data structures to map its input information into the DFHWS-DATA container, and to interpret any resulting response data. The appropriate binding file is used to transform the data format between that found in this container and that used within the messages that flow to and from the remote Web service.

Configuring CICS as a Web services requester

A second CICS region is used in the role of the Web services requester. As with the first region it requires the group DFHSTAND to be installed. It makes use of a PIPELINE resource to send Web service requests into the network. A TCPIPSERVICE resource is not needed by a requester pipeline, which instead makes use of the CICS sockets interface to send messages into the network and await message responses.

The PIPELINE, which on my system was called PIPEA1R, made use of the configuration file shown in Example 7, and was stored in directory number 1. The PIPELINE had its own SHELF and WSDIR directories defined in the Unix filesystem with the following names:

- */u/myfile/cicsts/webservices/shelf/requester1/*
- */u/myfile/cicsts/webservices/wsdir/requester1/*.

The binding file that this pipeline used was generated by the application using the JCL shown in Example 5. Here the program DFHWS2LS, which CICS provides in its SDFHINST library and which was copied to a working directory, is called to generate a binding file from some WSDL. The WSDL was produced when Example 4 was run. The binding file is placed in directory number 5.

Two other objects are created when this job is run. An input and an output copy book, with the names provided by the REQMEM and RESPMEM parameters, are placed in the library named by the PDSLIB. The program language they use is indicated by the LANG parameter. They are intended for inclusion in any program that invokes this Web service. However, in this simple test case they were not used.

Example 3 is the test application that was used to invoke the Web service. It contains a structure describing the COMMAREA used by the Web service's business logic program Example1. The instance of the structure in Example 3 is then populated with a four-character field, representing an integer value.

The structure is copied to the DFHWS-DATA container, which is in a named channel. The Web service is then invoked, passing this channel on the call. When the Web service responds, the program checks the response code and issues one or more messages indicating the success or failure of the operation.

A change needs to be made to the WS-URI string that is defined in the working storage section of Example 3. This must match the IP address used by your location, together with the port number used by the TCPIPSERVICE in the Web service provider region. Also the URI that follows the port number must be the same as that of the URI parameter in Example 4.

Once this has been done, the code can be compiled and linked into a library where the requester CICS region can find it. A program definition needs to be created together with a transaction, which can then be used to run this program. The transaction name I used was ADD1.

Next install the PIPELINE resource and look at the messages that are generated in MSGUSR. If the binding file has been correctly interpreted you should see a DFHPI0901 message showing the Web service ADDONE00 being created during the scan of pipeline PIPEA1R. This should be followed by message DFHPI0910, indicating that this operation has succeeded. A DFHPI0704 message should then follow this, indicating that one Web service has been successfully created and that no failures were detected.

Testing these configurations

All that remains to do is to test the configuration. You can run the transaction ADD1 from your requester region directly and look for the WTO messages it produces, or you might instead choose to run it under EDF to follow the commands as they are issued from the ADD1TEST program.

This program places 0041 into the DFHWS-DATA container before calling the service. It issues an INVOKE_WEBSERVICE call and checks the response code. If the call succeeds, the content of this container is retrieved and should contain the characters 0042 (a number that has particular significance for readers of Douglas Adams novels).

Once you have this test working, you can go on to deploy more complex workloads.

FOLLOW-ON WORK

CICS provides some extensions to its Web services support, including that for Atomic Transactions. These will be discussed in a follow-on article, where the examples in this article will be extended to create and test a basic WSAT infrastructure.

EXAMPLES

Example 1 – business logic that is enabled as a Web service:

```
*****************************************************************
       IDENTIFICATION DIVISION.
```

```
        PROGRAM-ID. ADDONEØØ.
        ENVIRONMENT DIVISION.
        CONFIGURATION SECTION.
       *
        DATA DIVISION.
        WORKING-STORAGE SECTION.

       *---------------------------------------------------------------*
       * DEFINE & INITIALIZE INPUT PARAMETERS
       *---------------------------------------------------------------*
        Ø1 WS-WORKING-STORAGE.
           Ø3 WS-ADDR-DFHCOMMAREA      USAGE IS POINTER.
           Ø3 WS-MYFIELD.
             Ø5 WS-MYF1                   PIC X(4).
             Ø5 WS-MYF2 REDEFINES WS-MYF1.
                Ø7 WS-NUM1                PIC S9(8) COMP.
             Ø5 WS-F-SPACES               PIC X(5Ø) VALUE SPACES.

       *****************************************************************
       *    L I N K A G E     S E C T I O N
       *****************************************************************
        LINKAGE SECTION.
        Ø1  DFHCOMMAREA.
            Ø3  CA-MYFIELD              PIC X(4).

       ****************************************************************
       *    P R O C E D U R E S
       ****************************************************************
        PROCEDURE DIVISION.

        MAINLINE SECTION.

       * INITIALIZE WORKING STORAGE FOR INPUT DATA TO BE RECEIVED INTO
            SET  WS-ADDR-DFHCOMMAREA TO ADDRESS OF DFHCOMMAREA.
            MOVE CA-MYFIELD          TO WS-MYFIELD.
            ADD  1                   TO WS-NUM1.
            MOVE WS-MYFIELD          TO CA-MYFIELD.

            EXEC CICS RETURN END-EXEC.


        MAINLINE-EXIT.
            EXIT.
```

Example 2 – business logic COMMAREA:

```
            Ø3  CA-MYFIELD              PIC X(4).
```

Example 3 – Web service invocation logic:

```
 *****************************************************************
  IDENTIFICATION DIVISION.
  PROGRAM-ID. ADD1TEST.
  ENVIRONMENT DIVISION.
  CONFIGURATION SECTION.
*
  DATA DIVISION.
  WORKING-STORAGE SECTION.

 *---------------------------------------------------------------*
* DEFINE & INITIALIZE INPUT PARAMETERS
 *---------------------------------------------------------------*
  Ø1 WS-WORKING-STORAGE.
     Ø3 WS-CONSTANTS.
       Ø5 WS-WSERVICE   PIC X(32) VALUE 'addoneØØ'.
       Ø5 WS-CHANNEL    PIC X(16) VALUE 'ADDONE_CHANNEL  '.
       Ø5 WS-IN         PIC X(16) VALUE 'DFHWS-DATA      '.
       Ø5 WS-OUT        PIC X(16) VALUE 'DFHWS-DATA      '.
       Ø5 WS-OP         PIC X(255) VALUE 'addoneØØOperation'.
       Ø5 FILLER        PIC X(1).
       Ø5 WS-DATA       PIC X(4) VALUE 'ØØ41'.
       Ø5 WS-HELLO      PIC X(2Ø) VALUE 'ADD1TEST HAS STARTED'.
       Ø5 WS-BYE        PIC X(21) VALUE 'ADD1TEST HAS FINISHED'.
       Ø5 FILLER        PIC X(3).
       Ø5 WS-BAD        PIC X(24) VALUE
         'ADD1TEST HAS ENDED BADLY'.
       Ø5 WS-URI        PIC X(255) VALUE
         'http://my.local.ip.address:9185/myid/cicsts/add1'.
     Ø3 WS-VARIABLES.
       Ø5 WS-RESULT                 PIC X(4).

 *****************************************************************
*    L I N K A G E    S E C T I O N
 *****************************************************************
  LINKAGE SECTION.
 ****************************************************************
*    P R O C E D U R E S
 ****************************************************************
  PROCEDURE DIVISION.

  MAINLINE SECTION.

      EXEC CICS WRITE OPERATOR TEXT(WS-HELLO)
           TEXTLENGTH(LENGTH OF WS-HELLO)
      END-EXEC.
*
      EXEC CICS PUT CONTAINER(WS-IN) FROM(WS-DATA)
           CHANNEL(WS-CHANNEL)
           FLENGTH(LENGTH OF WS-DATA) BIT
      END-EXEC.
 *
      EXEC CICS INVOKE WEBSERVICE(WS-WSERVICE)
           CHANNEL(WS-CHANNEL)
           OPERATION(WS-OP) URI(WS-URI)
           NOHANDLE
      END-EXEC.
 *
      EVALUATE EIBRESP
         WHEN DFHRESP(NORMAL)
            EXEC CICS GET CONTAINER(WS-OUT) INTO(WS-RESULT)
               CHANNEL(WS-CHANNEL)
            END-EXEC,
 *
            EXEC CICS WRITE OPERATOR TEXT(WS-BYE)
               TEXTLENGTH(LENGTH OF WS-BYE)
            END-EXEC,
            EXEC CICS WRITE OPERATOR TEXT(WS-RESULT)
               TEXTLENGTH(LENGTH OF WS-RESULT)
            END-EXEC,
 *
         WHEN OTHER
            EXEC CICS WRITE OPERATOR TEXT(WS-BAD)
               TEXTLENGTH(LENGTH OF WS-BAD)
            END-EXEC,
      END-EVALUATE.
 *
 *
      EXEC CICS RETURN END-EXEC.


  MAINLINE-EXIT.
      EXIT.
```

Example 4 – JCL to run the DFHLS2WS program:

```
//LS2WS JOB ,CLASS=A,NOTIFY=#######,MSGCLASS=A,REGION=ØM
//  SET QT=''''
//JCLLIB   JCLLIB ORDER=MYID.JCL.LIBRARY
//JAVAPROG EXEC DFHLS2WS,
// TMPFILE='MDB',
// JAVADIR='java142s/J1.4',
// USSDIR=''
//INPUT.SYSUT1 DD *
LOGFILE=/u/myfile/cicts/webservices/ws.log
PGMNAME=ADDONEØØ
URI=/myid/cicsts/add1
PGMINT=COMMAREA
LANG=COBOL
WSDL=/u/myid/cicsts/webservices/wsdir/provider1/addoneØØ.wsdl
WSBIND=/u/myid/cicts/webservices/wsdir/provider1/addoneØØ.wsbind
```

```
PDSLIB=//MYID.CICS.SOURCE
REQMEM=ADD1COMM
RESPMEM=ADD1COMM
/*
```

Example 5 – JCL to run the DFHWS2LS program:

```
//LS2WS JOB ,CLASS=A,NOTIFY=#######,MSGCLASS=A,REGION=ØM
//  SET QT=''''
//JCLLIB   JCLLIB ORDER=MYID.JCL.LIBRARY
//JAVAPROG EXEC DFHWS2LS,
// TMPFILE='MB',
// JAVADIR='java142s/J1.4',
// USSDIR=''
//INPUT.SYSUT1 DD *
LOGFILE=/u/myid/cicts/webservices/ws.log
WSDL=/u/myid/cicsts/webservices/wsdir/provider1/addoneØØ.wsdl
BINDING=ADDONEØØHTTPSoapBinding
LANG=COBOL
WSBIND=/u/myid/cicts/webservices/wsdir/provider1/addoneØØ.wsbind
PDSLIB=//MYID.CICS.SOURCE
REQMEM=A1REQ
RESPMEM=A1RESP
/*
```

Example 6 – configuration file for the provider pipeline:

```
<?xml version="1.Ø" encoding="UTF-8"?>
<provider_pipeline xmlns="http://www.ibm.com/software/htp/cics/pipeline"
 xmlns:xsi="http://www.w3.org/2ØØ1/XMLSchema-instance"
 xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
provider.xsd ">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler/>
    </terminal_handler>
  </service>
   <apphandler>DFHPITP</apphandler>
</provider_pipeline>
```

Example 7 – configuration file for the requester pipline:

```
<?xml version="1.Ø" encoding="UTF-8"?>
<requester_pipeline xmlns="http://www.ibm.com/software/htp/cics/
pipeline"
   xmlns:xsi="http://www.w3.org/2ØØ1/XMLSchema-instance"
   xsi:schemaLocation="http://www.ibm.com/software/htp/cics/pipeline
requester.xsd">
  <service>
    <service_handler_list>
```

```
      <cics_soap_1.1_handler/>
    </service_handler_list>
  </service>
</requester_pipeline>
```

Michael D Brooks
Advisory Software Engineer
IBM (UK)

BigCommarea utility user guide – part 2

This month we conclude the COBOL code for a simple utility highlighting BigCommarea usage (BTS data containers).

```
028400
028500              IF BTS-RESP1 = DFHRESP(NOTAUTH)
028600                  SET USER-NOT-AUTH-ON-REPO-FILE TO TRUE
028700              END-IF
028800
028900              IF BTS-RESP1 = DFHRESP(IOERR)
029000                  IF BTS-RESP2 = 29
029100                      SET REPO-FILE-NOT-AVAILABLE TO TRUE
029200                  END-IF
029300                  IF BTS-RESP2 = 30
029400                      SET IO-ERROR-ON-REPOSITORY-FILE TO TRUE
029500                  END-IF
029600              END-IF
029700
029800              IF BTS-RESP1 = DFHRESP(LOCKED)
029900                  SET RECORD-LOCKED-IN-REPO-FILE TO TRUE
030000              END-IF
030100
030200              IF BTS-RESP1 = DFHRESP(PROCESSBUSY)
030300                  SET REQUEST-TIMED-OUT TO TRUE
030400              END-IF
030500
030600              IF BTS-RESP1 = DFHRESP(PROCESSERR)
030700                  IF BTS-RESP2 = 09
030800                      SET PREOCESS-TYPE-DOES-NOT-EXSIT TO TRUE
030900                  END-IF
031000                  IF BTS-RESP2 = 14
031100                      SET OTHER-ERROR TO TRUE
031200                  END-IF
```

```
             END-IF
         END-IF
     END-IF
     .

BØØØ-ACQUIRE-PROCESS.
    SET BTS-ACQUIRE TO TRUE
    EXEC CICS ACQUIRE
        PROCESS(BTS-CONTAINER-MGR-NAME)
        PROCESSTYPE(BTS-PROCESS-TYPE)
        RESP(BTS-RESP1)
        RESP2(BTS-RESP2)
    END-EXEC
    IF BTS-RESP1 = DFHRESP(NORMAL)
        OR BTS-RESP1 = DFHRESP(INVREQ)
        CONTINUE
    ELSE
        SET BTS-FAILURE TO TRUE
        SET OTHER-ERROR TO TRUE

        IF BTS-RESP1 = DFHRESP(NOTAUTH)
            SET USER-NOT-AUTH-ON-REPO-FILE TO TRUE
        END-IF

        IF BTS-RESP1 = DFHRESP(IOERR)
            IF BTS-RESP2 = 29
                SET REPO-FILE-NOT-AVAILABLE TO TRUE
            END-IF
            IF BTS-RESP2 = 3Ø
                SET IO-ERROR-ON-REPOSITORY-FILE TO TRUE
            END-IF
        END-IF

        IF BTS-RESP1 = DFHRESP(LOCKED)
            SET RECORD-LOCKED-IN-REPO-FILE TO TRUE
        END-IF

        IF BTS-RESP1 = DFHRESP(PROCESSBUSY)
            OR BTS-RESP1 = DFHRESP(ACTIVITYBUSY)
            SET REQUEST-TIMED-OUT TO TRUE
        END-IF

        IF BTS-RESP1 = DFHRESP(PROCESSERR)
            IF BTS-RESP2 = Ø5
                SET PROCESS-DOES-NOT-EXIST TO TRUE
            END-IF
            IF BTS-RESP2 = Ø9
                SET PREOCESS-TYPE-DOES-NOT-EXSIT TO TRUE
            END-IF
        END-IF
```

```
036300     END-IF
036400     .
036500*
036600 B100-CHECK-FILE-STATUS.
036700     EXEC CICS INQUIRE
036710         PROCESSTYPE(BTS-PROCESS-TYPE)
036720         FILE(BTS-FILE-NAME)
036730     END-EXEC
036800     EXEC CICS INQUIRE
036900         FILE (BTS-FILE-NAME)
037000         OPENSTATUS(FILE-OPEN-STATUS)
037100     END-EXEC
037200     IF  FILE-OPEN-STATUS = DFHVALUE (OPEN)
037300         CONTINUE
037500     ELSE
037600         SET BTS-FAILURE TO TRUE
037610         SET REPO-FILE-NOT-AVAILABLE TO TRUE
037800     END-IF
037900     .
```

BTSSAMP1

```
000100*****************************************************************
000200*           THIS PROGRAM DEVELOPED FOR TESTING                  *
000300*     BTS BIGCOMMAREA PROCESS : USE CEDF TO EXECUTE             *
000400*                                                               *
000500*   RELATED PROGRAMS -  BTSUTIL1                                *
000600*   GROUP            -  GENGRP1, CICS1 REGION                   *
000700*   PROCESS GROUP    -  GENUSER                                 *
000800*   PROCESSTYPE      -  BIGCOMM                                 *
000900*   VSAM FILE NAME   -  BIGCOMM                                 *
001000*   PROGRAM NAME     -  BTSSAMP1                                *
001100*   MAPS USED        -  NONE                                    *
001200*   TRANSACTION ID   -  BT01                                    *
001300*                                                               *
001400*****************************************************************
001500 IDENTIFICATION DIVISION.
001600 PROGRAM-ID. BTSSAMP1.
001700 AUTHOR. SARMAKVRS.
001800 DATA DIVISION.
001900 WORKING-STORAGE SECTION.
002000 01  BTS-COPY-BOOK.
002100     COPY  BTSCOPY1.
002200 01  BTS-PROG                     PIC   X(08) VALUE 'BTSUTIL1'.
002300 01  BTS-DATA-AREA                PIC   X(01) VALUE SPACES.
002400*
002500 01  MSG-OUT                      PIC   X(50) VALUE SPACES.
002600 01  RESP1-VAR                    PIC   S9(08) COMP.
002700 01  RESP2-VAR                    PIC   S9(08) COMP.
```

```
002800*
002900 PROCEDURE DIVISION.
003000 A000-MAIN-PARA.
003100     MOVE 'TESTING BTS'            TO  MSG-OUT
003110     PERFORM B100-DISPLAY-TEXT
003600
003700     MOVE 'PROCESS1'               TO  BTS-CONTAINER-MGR-NAME
003800     MOVE 'BIGCOMM'                TO  BTS-PROCESS-TYPE
003900     SET  BTS-CREATE-CONTAINER-MGR TO  TRUE
004000
004100     CALL BTS-PROG  USING DFHEIBLK, DFHCOMMAREA,
004200                          BTS-COPY-BOOK, BTS-DATA-AREA
004300     IF  BTS-SUCCESSFUL
004400         MOVE 'BTS SUCCESS'            TO  MSG-OUT
004410         PERFORM B100-DISPLAY-TEXT
004900     ELSE
005000         MOVE 'BTS FAILED '            TO  MSG-OUT
005010         PERFORM B100-DISPLAY-TEXT
005020         PERFORM B000-ERROR-HANLDING-PARA
005500     END-IF
005600     EXEC CICS SYNCPOINT
005700          RESP(RESP1-VAR)
005800          RESP2(RESP2-VAR)
005900     END-EXEC
006000     EXEC CICS RETURN
006100     END-EXEC
006200     .
006210
006211 B000-ERROR-HANLDING-PARA.
006220     EVALUATE TRUE
006300         WHEN  NO-FAILURE
006310             MOVE  'NO FAILURE'  TO  MSG-OUT
006400         WHEN  PREOCESS-TYPE-DOES-NOT-EXSIT
006500             MOVE  'PREOCESS TYPE DOES NOT EXSIT'  TO MSG-OUT
006600         WHEN  PROCESS-DOES-NOT-EXIST
006610             MOVE  'CONTAINER MGR DOES NOT EXIST'  TO MSG-OUT
006700         WHEN  DATA-LENGTH-MISMATCH
006710             MOVE  'DATA LENGTH MISMATCH'  TO MSG-OUT
006800         WHEN  PROCESS-BUSY
006810             MOVE  'PROCESS BUSY'  TO MSG-OUT
006900         WHEN  CONTAINER-DOES-NOT-EXIST
006910             MOVE  'CONTAINER DOES NOT EXIST'  TO MSG-OUT
007000         WHEN  REPO-FILE-NOT-AVAILABLE
007010             MOVE  'REPO FILE NOT AVAILABLE'  TO MSG-OUT
007100         WHEN  IO-ERROR-ON-REPOSITORY-FILE
007200             MOVE  'IO ERROR ON REPOSITORY FILE'  TO MSG-OUT
007300         WHEN  RECORD-IN-REPO-FILE-IN-USE
007400             MOVE  'RECORD IN REPO FILE IN USE'  TO MSG-OUT
007500         WHEN  RECORD-LOCKED-IN-REPO-FILE
007600             MOVE  'RECORD LOCKED IN REPO FILE'  TO MSG-OUT
```

```
007700         WHEN  USER-NOT-AUTH-ON-REPO-FILE
007800             MOVE  'USER NOT AUTH ON REPO FILE'  TO MSG-OUT
007900         WHEN  REQUEST-TIMED-OUT
007910             MOVE  'REQUEST TIMED OUT'  TO MSG-OUT
008000         WHEN  INVALID-BTS-REQUEST
008010             MOVE  'INVALID REQUEST'  TO MSG-OUT
008100         WHEN  UNKNOWN-BTS-REQUEST
008110             MOVE  'UNKNOWN REQUEST'  TO MSG-OUT
008200         WHEN  USER-NOT-AUTHORIZED
008210             MOVE  'USER NOT AUTHORIZED'  TO MSG-OUT
008300         WHEN  PROCESS-ALREADY-IN-USE
008310             MOVE  'PROCESS NAME ALREADY IN USE'  TO MSG-OUT
008400         WHEN  PROCESSTYPE-ALREADY-IN-USE
008500             MOVE  'PROCESSTYPE ALREADY IN USE'  TO MSG-OUT
008600         WHEN  TRANSID-CAN-NOT-BE-FOUND
008610             MOVE  'TRANSID CAN NOT BE FOUND'  TO MSG-OUT
008700         WHEN  PROCESS-NOT-ENABLED
008710             MOVE  'PROCESS NOT ENABLED'  TO MSG-OUT
008800         WHEN  OTHER-ERROR
008900             MOVE  'OTHER ERROR'  TO MSG-OUT
009000     END-EVALUATE
009001     PERFORM B100-DISPLAY-TEXT
009002     .
009003 B100-DISPLAY-TEXT.
009010     EXEC CICS SEND TEXT
009020         FROM (MSG-OUT)
009030         LENGTH (LENGTH OF MSG-OUT)
009040     END-EXEC
009100     .
```

BTSSAMP2

```
000100***************************************************************
000200*           THIS PROGRAM DEVELOPED FOR TESTING                *
000300*     BTS BIGCOMMAREA PROCESS : USE CEDF TO EXECUTE           *
000400*                                                             *
000500*   RELATED PROGRAMS -  BTSUTIL1                              *
000600*   GROUP            -  GENGRP1, CICS1 REGION                 *
000700*   PROCESS GROUP    -  GENUSER                               *
000800*   PROCESSTYPE      -  BIGCOMM                               *
000900*   VSAM FILE NAME   -  BIGCOMM                               *
001000*   PROGRAM NAME     -  BTSSAMP2                              *
001100*   MAPS USED        -  NONE                                  *
001200*   TRANSACTION ID   -  BT02                                  *
001300*                                                             *
001400***************************************************************
001500 IDENTIFICATION DIVISION.
001600 PROGRAM-ID. BTSSAMP2.
001700 AUTHOR. SARMAKVRS.
```

```
001800 DATA DIVISION.
001900 WORKING-STORAGE SECTION.
002000 01  BTS-COPY-BOOK.
002100     COPY  BTSCOPY1.
002200 01  BTS-PROG                    PIC   X(08) VALUE 'BTSUTIL1'.
002300 01  BTS-DATA-AREA               PIC   X(01) VALUE SPACE.
002400 01  DATA-VAR                    PIC   X(50)  VALUE SPACES.
002500*
002600 01  MSG-OUT                     PIC   X(50) VALUE SPACES.
002700 01  RESP1-VAR                   PIC   S9(08) COMP.
002800 01  RESP2-VAR                   PIC   S9(08) COMP.
002900*
003000 PROCEDURE DIVISION.
003100 A000-MAIN-PARA.
003200     MOVE 'TESTING BTS'             TO  MSG-OUT
003210     PERFORM B100-DISPLAY-TEXT
003700
003800     MOVE 'PROCESS1'                TO  BTS-CONTAINER-MGR-NAME
003900     MOVE 'BIGCOMM'                 TO  BTS-PROCESS-TYPE
004000     MOVE 'CONTAINER1'              TO  BTS-CONTAINER-NAME
004100     SET  BTS-CREATE-CONTAINER      TO  TRUE
004200
004300     CALL BTS-PROG  USING DFHEIBLK, DFHCOMMAREA,
004400                          BTS-COPY-BOOK, BTS-DATA-AREA
004500     IF  BTS-SUCCESSFUL
004600         MOVE 'CREATE SUCCESS'         TO  MSG-OUT
004610         PERFORM B100-DISPLAY-TEXT
005100     ELSE
005200         MOVE 'CREATE FAILED'           TO  MSG-OUT
005210         PERFORM B100-DISPLAY-TEXT
005220         PERFORM B000-ERROR-HANLDING-PARA
005700     END-IF
006200*
006300     MOVE 'THIS IS MY DATA'         TO  DATA-VAR
006400*
006500     MOVE +50                       TO  BTS-DATA-LENGTH
006600     MOVE 'CONTAINER1'              TO  BTS-CONTAINER-NAME
006700     MOVE 'PROCESS1'                TO  BTS-CONTAINER-MGR-NAME
006800     MOVE 'BIGCOMM'                 TO  BTS-PROCESS-TYPE
006900     SET  BTS-PUT-TO-CONTAINER      TO  TRUE
007000
007100     CALL BTS-PROG  USING DFHEIBLK, DFHCOMMAREA,
007200                          BTS-COPY-BOOK, DATA-VAR
007300*
007400     IF  BTS-SUCCESSFUL
007500         MOVE 'PUT SUCCESS'             TO  MSG-OUT
007510         PERFORM B100-DISPLAY-TEXT
008000     ELSE
008100         MOVE 'PUT FAILED '             TO  MSG-OUT
008110         PERFORM B100-DISPLAY-TEXT
```

```
008120          PERFORM B000-ERROR-HANLDING-PARA
008600      END-IF
009100*
009200      MOVE SPACES                   TO  DATA-VAR
009300*
009400      MOVE +50                      TO  BTS-DATA-LENGTH
009500      MOVE 'CONTAINER1'             TO  BTS-CONTAINER-NAME
009600      MOVE 'PROCESS1'               TO  BTS-CONTAINER-MGR-NAME
009700      MOVE 'BIGCOMM'                TO  BTS-PROCESS-TYPE
009800      SET  BTS-GET-FROM-CONTAINER   TO  TRUE
009900
010000      CALL BTS-PROG  USING DFHEIBLK, DFHCOMMAREA,
010100                           BTS-COPY-BOOK, DATA-VAR
010200*
010300      IF  BTS-SUCCESSFUL
010400          MOVE 'GET SUCCESS'            TO  MSG-OUT
010410          PERFORM B100-DISPLAY-TEXT
010900      ELSE
011000          MOVE 'GET FAILED '            TO  MSG-OUT
011010          PERFORM B100-DISPLAY-TEXT
011020          PERFORM B000-ERROR-HANLDING-PARA
011500      END-IF
012000*
012100      MOVE 'CONTAINER1'             TO  BTS-CONTAINER-NAME
012200      MOVE 'PROCESS1'               TO  BTS-CONTAINER-MGR-NAME
012300      MOVE 'BIGCOMM'                TO  BTS-PROCESS-TYPE
012400      SET  BTS-DELETE-CONTAINER     TO  TRUE
012500
012600      CALL BTS-PROG  USING DFHEIBLK, DFHCOMMAREA,
012700                           BTS-COPY-BOOK, BTS-DATA-AREA
012800*
012900      IF  BTS-SUCCESSFUL
013000          MOVE 'DELETE SUCCESS'            TO  MSG-OUT
013010          PERFORM B100-DISPLAY-TEXT
013500      ELSE
013600          MOVE 'DELETE FAILED '            TO  MSG-OUT
013610          PERFORM B100-DISPLAY-TEXT
013620          PERFORM B000-ERROR-HANLDING-PARA
014100      END-IF
014200*
014210      PERFORM B100-DISPLAY-TEXT
014700      EXEC CICS SYNCPOINT
014800           RESP(RESP1-VAR)
014900           RESP2(RESP2-VAR)
015000      END-EXEC
015100      EXEC CICS RETURN
015200      END-EXEC
015300      .
015310*
015400  B000-ERROR-HANLDING-PARA.
```

```
           EVALUATE TRUE
               WHEN  NO-FAILURE
                   MOVE  'NO FAILURE'  TO  MSG-OUT
               WHEN  PREOCESS-TYPE-DOES-NOT-EXSIT
                   MOVE  'PREOCESS TYPE DOES NOT EXSIT'  TO MSG-OUT
               WHEN  PROCESS-DOES-NOT-EXIST
                   MOVE  'CONTAINER MGR DOES NOT EXIST'  TO MSG-OUT
               WHEN  DATA-LENGTH-MISMATCH
                   MOVE  'DATA LENGTH MISMATCH'  TO MSG-OUT
               WHEN  PROCESS-BUSY
                   MOVE  'PROCESS BUSY'  TO MSG-OUT
               WHEN  CONTAINER-DOES-NOT-EXIST
                   MOVE  'CONTAINER DOES NOT EXIST'  TO MSG-OUT
               WHEN  REPO-FILE-NOT-AVAILABLE
                   MOVE  'REPO FILE NOT AVAILABLE'  TO MSG-OUT
               WHEN  IO-ERROR-ON-REPOSITORY-FILE
                   MOVE  'IO ERROR ON REPOSITORY FILE'  TO MSG-OUT
               WHEN  RECORD-IN-REPO-FILE-IN-USE
                   MOVE  'RECORD IN REPO FILE IN USE'  TO MSG-OUT
               WHEN  RECORD-LOCKED-IN-REPO-FILE
                   MOVE  'RECORD LOCKED IN REPO FILE'  TO MSG-OUT
               WHEN  USER-NOT-AUTH-ON-REPO-FILE
                   MOVE  'USER NOT AUTH ON REPO FILE'  TO MSG-OUT
               WHEN  REQUEST-TIMED-OUT
                   MOVE  'REQUEST TIMED OUT'  TO MSG-OUT
               WHEN  INVALID-BTS-REQUEST
                   MOVE  'INVALID REQUEST'  TO MSG-OUT
               WHEN  UNKNOWN-BTS-REQUEST
                   MOVE  'UNKNOWN REQUEST'  TO MSG-OUT
               WHEN  USER-NOT-AUTHORIZED
                   MOVE  'USER NOT AUTHORIZED'  TO MSG-OUT
               WHEN  PROCESS-ALREADY-IN-USE
                   MOVE  'PROCESS NAME ALREADY IN USE'  TO MSG-OUT
               WHEN  PROCESSTYPE-ALREADY-IN-USE
                   MOVE  'PROCESSTYPE ALREADY IN USE'  TO MSG-OUT
               WHEN  TRANSID-CAN-NOT-BE-FOUND
                   MOVE  'TRANSID CAN NOT BE FOUND'  TO MSG-OUT
               WHEN  PROCESS-NOT-ENABLED
                   MOVE  'PROCESS NOT ENABLED'  TO MSG-OUT
               WHEN  OTHER-ERROR
                   MOVE  'OTHER ERROR'  TO MSG-OUT
           END-EVALUATE
           PERFORM B100-DISPLAY-TEXT
           .
       B100-DISPLAY-TEXT.
           EXEC CICS SEND TEXT
               FROM (MSG-OUT)
               LENGTH (LENGTH OF MSG-OUT)
           END-EXEC
           .
```

BTSSAMP3

```
***************************************************************
*           THIS PROGRAM DEVELOPED FOR TESTING                *
*     BTS BIGCOMMAREA PROCESS : USE CEDF TO EXECUTE           *
*                                                             *
*   RELATED PROGRAMS -  BTSUTIL1                              *
*   GROUP            -  GENGRP1, CICS1 REGION                 *
*   PROCESS GROUP    -  GENUSER                               *
*   PROCESSTYPE      -  BIGCOMM                               *
*   VSAM FILE NAME   -  BIGCOMM                               *
*   PROGRAM NAME     -  BTSSAMP3                              *
*   MAPS USED        -  NONE                                  *
*   TRANSACTION ID   -  BT03                                  *
*                                                             *
***************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID. BTSSAMP3.
 AUTHOR. SARMAKVRS.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
 01  BTS-COPY-BOOK.
     COPY  BTSCOPY1.
 01  BTS-PROG                   PIC   X(08) VALUE 'BTSUTIL1'.
 01  BTS-DATA-AREA              PIC   X(01) VALUE SPACES.
*
 01  MSG-OUT                    PIC   X(50) VALUE SPACES.
 01  RESP1-VAR                  PIC   S9(08) COMP.
 01  RESP2-VAR                  PIC   S9(08) COMP.
*
 PROCEDURE DIVISION.
 A000-MAIN-PARA.
     MOVE 'TESTING BTS'            TO  MSG-OUT
     PERFORM B100-DISPLAY-TEXT

     MOVE 'PROCESS1'               TO  BTS-CONTAINER-MGR-NAME
     MOVE 'BIGCOMM'                TO  BTS-PROCESS-TYPE
     SET  BTS-DELETE-CONTAINER-MGR TO  TRUE

     CALL BTS-PROG  USING DFHEIBLK, DFHCOMMAREA,
                          BTS-COPY-BOOK, BTS-DATA-AREA
     IF  BTS-SUCCESSFUL
         MOVE 'BTS SUCCESS'            TO  MSG-OUT
         PERFORM B100-DISPLAY-TEXT
     ELSE
         MOVE 'BTS FAILED '            TO  MSG-OUT
         PERFORM B100-DISPLAY-TEXT
         PERFORM B000-ERROR-HANLDING-PARA
     END-IF
     EXEC CICS SYNCPOINT
```

```
005700          RESP(RESP1-VAR)
005800          RESP2(RESP2-VAR)
005900     END-EXEC
006000     EXEC CICS RETURN
006100     END-EXEC
006200     .
006300
006400 B000-ERROR-HANLDING-PARA.
006500     EVALUATE TRUE
006600         WHEN  NO-FAILURE
006700             MOVE  'NO FAILURE'  TO  MSG-OUT
006800         WHEN  PREOCESS-TYPE-DOES-NOT-EXSIT
006900             MOVE  'PREOCESS TYPE DOES NOT EXSIT'  TO MSG-OUT
007000         WHEN  PROCESS-DOES-NOT-EXIST
007100             MOVE  'CONTAINER MGR DOES NOT EXIST'  TO MSG-OUT
007200         WHEN  DATA-LENGTH-MISMATCH
007300             MOVE  'DATA LENGTH MISMATCH'  TO MSG-OUT
007400         WHEN  PROCESS-BUSY
007500             MOVE  'PROCESS BUSY'  TO MSG-OUT
007600         WHEN  CONTAINER-DOES-NOT-EXIST
007700             MOVE  'CONTAINER DOES NOT EXIST'  TO MSG-OUT
007800         WHEN  REPO-FILE-NOT-AVAILABLE
007900             MOVE  'REPO FILE NOT AVAILABLE'  TO MSG-OUT
008000         WHEN  IO-ERROR-ON-REPOSITORY-FILE
008100             MOVE  'IO ERROR ON REPOSITORY FILE'  TO MSG-OUT
008200         WHEN  RECORD-IN-REPO-FILE-IN-USE
008300             MOVE  'RECORD IN REPO FILE IN USE'  TO MSG-OUT
008400         WHEN  RECORD-LOCKED-IN-REPO-FILE
008500             MOVE  'RECORD LOCKED IN REPO FILE'  TO MSG-OUT
008600         WHEN  USER-NOT-AUTH-ON-REPO-FILE
008700             MOVE  'USER NOT AUTH ON REPO FILE'  TO MSG-OUT
008800         WHEN  REQUEST-TIMED-OUT
008900             MOVE  'REQUEST TIMED OUT'  TO MSG-OUT
009000         WHEN  INVALID-BTS-REQUEST
009100             MOVE  'INVALID REQUEST'  TO MSG-OUT
009200         WHEN  UNKNOWN-BTS-REQUEST
009300             MOVE  'UNKNOWN REQUEST'  TO MSG-OUT
009400         WHEN  USER-NOT-AUTHORIZED
009500             MOVE  'USER NOT AUTHORIZED'  TO MSG-OUT
009600         WHEN  PROCESS-ALREADY-IN-USE
009700             MOVE  'PROCESS NAME ALREADY IN USE'  TO MSG-OUT
009800         WHEN  PROCESSTYPE-ALREADY-IN-USE
009900             MOVE  'PROCESSTYPE ALREADY IN USE'  TO MSG-OUT
010000         WHEN  TRANSID-CAN-NOT-BE-FOUND
010100             MOVE  'TRANSID CAN NOT BE FOUND'  TO MSG-OUT
010200         WHEN  PROCESS-NOT-ENABLED
010300             MOVE  'PROCESS NOT ENABLED'  TO MSG-OUT
010400         WHEN  OTHER-ERROR
010500             MOVE  'OTHER ERROR'  TO MSG-OUT
010600     END-EVALUATE
```

```
           PERFORM B1ØØ-DISPLAY-TEXT
           .
       B1ØØ-DISPLAY-TEXT.
           EXEC CICS SEND TEXT
               FROM (MSG-OUT)
               LENGTH (LENGTH OF MSG-OUT)
           END-EXEC
           .
```

BTS APIs

If you decide not to use the BTS utility, you may use the following BTS APIs in your program:

1 **EXEC CICS DEFINE PROCESS** – define or create a new process.

2 **EXEC CICS CANCEL ACQPROCESS** – delete an existing process.

3 **EXEC CICS ACQUIRE PROCESS** – connect to an existing process.

4 **EXEC CICS PUT CONTAINER** – put data into a container.

5 **EXEC CICS GET CONATINER** – get data from a container.

6 **EXEC CICS DELETE CONTAINER** – delete a container.

Define process

Containers store data. Without having a process, it is not possible to create and use any containers. Hence the first thing is to create a process.

The command is:

```
MOVE    'BIGCOMM'        TO    BTS-PROCESS-TYPE
MOVE    'PROCESS1'       TO    BTS-CONTAINER-MGR-NAME
EXEC CICS DEFINE
        PROCESS(BTS-CONTAINER-MGR-NAME)
        PROCESSTYPE(BTS-PROCESS-TYPE)
        TRANSID(EIBTRNID)
        RESP(RESP1-VAR)
        RESP2(RESP2-VAR)
END-EXEC
```

This command creates a process of BTS-PROCESS-TYPE with the name BTS-CONTAINER-MGR-NAME.

Note: if these processes are used as an actual process, the transaction ID used in the TRANSID option will have an important role. However, in this case the process will never be started and hence the transaction ID does not have any role to play. So, simply use EIBTRNID, which takes the current transaction's transaction ID.

Acquire process

When a unit-of-work ends, ie when a commit or rollback occurs, the process gets disconnected. Hence in every new unit-of-work, it is necessary to re-connect to the process.

The command is:

```
MOVE   'BIGCOMM'        TO   BTS-PROCESS-TYPE
MOVE   'PROCESS1'       TO   BTS-CONTAINER-MGR-NAME
EXEC CICS ACQUIRE
        PROCESS(BTS-CONTAINER-MGR-NAME)
        PROCESSTYPE(BTS-PROCESS-TYPE)
        RESP(RESP1-VAR)
        RESP2(RESP2-VAR)
END-EXEC
```

Delete process

The command to delete the process that is currently acquired is:

```
EXEC CICS CANCEL
        ACQPROCESS
        RESP(RESP1-VAR)
        RESP2(RESP2-VAR)
END-EXEC
```

Put container

If the container does not exist, the **put container** command creates a new container and writes data into it. If a container already exists with that name, it overwrites the container data. Hence this command can be seen as an insert/update command. (It is important to note that the **put container** command completely replaces old data with the new data.)

The command is:

```
MOVE    'CONATAINER1'     TO   BTS-CONTAINER-NAME
MOVE    +2Ø               TO   BTS-DATA-LENGTH
MOVE    'THIS IS MY INPUT DATA'
                          TO   BTS-DATA-BUFFER
EXEC CICS PUT
   CONTAINER(BTS-CONTAINER-NAME)
   ACQPROCESS
   FROM(BTS-DATA-BUFFER)
   FLENGTH(BTS-DATA-LENGTH)
   RESP(RESP1-VAR)
   RESP2(RESP2-VAR)
END-EXEC
```

In the above command, a container name, length of the data buffer, and a buffer containing data are given.

Get container

The **get container** command reads data from a container. However, it is important to note that the whole data has to be read in one go, and hence you should know the exact length ahead of time. Data will not be removed from the container on a get command. Hence the same data can be read any number of times, before it is deleted.

The command is:

```
MOVE    'CONATAINER1'     TO   BTS-CONTAINER-NAME
MOVE    +2Ø               TO   BTS-DATA-LENGTH
MOVE    SPACES            TO   BTS-DATA-BUFFER
EXEC CICS GET
    CONTAINER(BTS-CONTAINER-NAME)
    ACQPROCESS
    INTO(BTS-DATA-BUFFER)
    FLENGTH(BTS-DATA-LENGTH)
    RESP(RESP1-VAR)
    RESP2(RESP2-VAR)
END-EXEC
```

Note: a GET operation with FLENGTH = 0 will get the length. If you do not know the exact length of data in the container, this method can be used.

Delete container

The **delete container** command, deletes a container and releases all the resources held by it.

The command is:

```
MOVE    'CONATAINER1'     TO   BTS-CONTAINER-NAME
EXEC CICS DELETE
        CONTAINER(BTS-CONTAINER-NAME)
        ACQPROCESS
        RESP(RESP1-VAR)
        RESP2(RESP2-VAR)
END-EXEC
```

Notes and tips:

1 When a commit/rollback occurs, the process gets disconnected. Hence after a commit/rollback it is necessary to re-connect using the **acquire process** command.

2 All BTS commands are hardened at commit only.

3 Data in a container can be read by any transaction that knows the process type, process, and container name.

4 Within a single process, there can be multiple containers with different names.

5 Across the processes, containers with the same name can exist.

6 It is not possible to append the contents in a container. All the data in a container has to be read or written in a single operation.

7 A read operation does not destroy the data. It can be read multiple times.

8 The PUT operation creates a container if it does not exist. If the container already exists, data in the container is over-written. (Old data is removed and new data is inserted.)

9 A GET operation with FLENGTH = 0 will get the length. If you do not know the exact length of data in the container, this method can be used.

10 A PUT with FLENGTH = 0 will remove the container data and release all resources. However, the container will not be deleted. The container can be used later to put data.

11 A CBAM transaction can be used to identify the process types defined in the CICS system. The same transaction gives details about the repository file and processes existing within each process. This transaction can be very useful in cases of failure of BTS programs.

12 Sample RDO entry details and administration information is available from the first reference document mentioned below.

REFERENCES

1 BigCommareas: *How to Bypass the 32K Commarea Restriction in CICS Transaction Server*. http://www-306.ibm.com/software/htp/cics/library/indexes/whitepapers.html. (It is a must-read by anyone who is going to use BigCommarea concepts.)

2 *CICS Business Transaction Services*, IBM manual.

3 *CICS Application Programming Guide*, IBM manual.

K V R S Sarma, K Viswanathan
Technical Architect, Systems Programmer
Infosys Technologies Ltd (India)

CICS questions and answers

Q I'd like to open up an HTTP port into CICS so that other mainframe applications (batch) can invoke my Web services via HTTP (instead of using EXCI). Security is a bit of an issue, Basic Authentication and SSL are not viable options. I'm a bit worried about opening up a port because it would be available to anyone to use. I've been told that relying on client IP addresses is unsafe. Is there a way to lock down the clients we expect?

A You're right, relying on client IP addresses can be unsafe. There are techniques that allow 'spoof' IP packets. It is possible to create an IP packet with an incorrect IP address for the client. You wouldn't get a reply (because it cannot route back to you), but the request would certainly reach its target and invoke whatever function is there. Fortunately, you can lock down a port to be accessible only by the local TCP/IP stack. If the port is bound to a certain specific IP address, 127.0.0.1, then only requests for that IP address from the local TCP/IP stack can get to it. This is because 127.0.0.1 always represents the local host, ie the mainframe, PC, Unix box, or NT server that you are currently on. If anyone not on the mainframe tries to access it, the request will just loop back to their own machine.

This would mean that you would need to have a CICS region for each LPAR you would run on. The advantage is that the IP address for each LPAR in this would be the same, so that would simplify coding, because 127.0.0.1 is always the local mainframe LPAR you are on. In other words, your batch application can use the Web service by sending a request to http://127.0.0.1: [cics port]/[Web Service program] and this port is available and 'seen' only by other mainframe applications on the same LPAR.

If you have any CICS-related questions, please send them in and we will do our best to find answers. Alternatively, e-mail them directly to cicsq@xephon.net.

© Xephon 2005

CICS news

IBM has announced Version 2.1 of the CICS Interdependency Analyzer for z/OS, which can be used with CICS TS to identify the resources used by CICS transactions and the relationships between them. The resources identified include those associated with transactions, programs, BMS maps, files, temporary storage queues, transient data queues, 3270 Bridge facility, Web Services, CorbaServer, and Enterprise JavaBeans (EJBs).

In addition, it creates reports on DB2, IMS, and WebSphere MQ resources used by CICS. This information can be used to maintain and enhance the performance of critical CICS applications. It also helps to speed CICS application migration and reuse, and increases CICS system availability.

Version 2.1 now includes a new Eclipse-based GUI, giving easy access to the resource relationship data in the database and improved query management facilities. This is based on an XML API. There's now timer-based collection, which allow the user to control when and in which CICS region the data collection is enabled.

Other enhancements include a program/transaction exclude list, ISPF customization of installation jobs, and a new flag on EXEC CICS START to show whether a REQID is present.

For further information contact:
URL: www.ibm.com/software/htp/cics/ianaly/support.

* * *

Rosebud Management Systems has announced Version 4.0 of Eden Server, its COBOL/CICS emulator.

The product is designed for mainframe decommissioning projects. Version 4.0 now provides direct access to all Eden based CICS transactions via the Web without the need for special programming or third-party software. It also provides full support for MVS JCL concepts and constructs such as in-line procedures, complete return code checking, as well as full support for batch utilities such as IDCAMS and SORT/MERGE.

For further information contact:
URL: www.rosebudusa.com/products.html.

* * *

Callataÿ & Wouters has announced a z/OS version of its THALER banking software. This version, called z/THALER, is designed to run under CICS and use DB2.

z/THALER can be installed on a zSeries platform and can easily interface with existing applications because of its native functioning CICS/DB2.

For further information contact:
URL: www.callatay-wouters.com/be/en/index.htm.

* * *

xephon