

# Enable atomic transaction support for web services in CICS

## An initial approach

Skill Level: Intermediate

[Oliver R. Fenton \(fentono@uk.ibm.com\)](mailto:fentono@uk.ibm.com)  
CICS Test Software Engineer  
EMC

29 Nov 2005

Learn about web services in IBM CICS Transaction Server for Z/OS Version 3.1 and how you can enable atomic transactions to provide commit and recovery functionality between applications across programming languages and platforms.

## Section 1. Before you start

In this tutorial you will install and run simple CICS web service provider and requester applications, before enabling atomic transactions and seeing a demonstration of the recovery facilities included in CICS.

### About this tutorial

Using a simple web service, you update a recoverable resource, showing the consequences of not enabling atomic transaction support between the provider and requester. By enabling web service atomic transactions and calling the same service, this exercise then demonstrates the ability to recover resources across systems using a set of coordination messages. The third section uses a header handler application to show the coordination message flows between the provider and requester systems. The final part of this tutorial explains how you can easily extend the infrastructure you use to interact with other web services-enabled

products.

## Objectives

Once complete you should be able to enable atomic transaction support for web services in CICS as a requester or provider and understand the SOAP message flows that take place between coordinator and participant.

Using this new functionality and the existing transactional features that CICS supports, you can create fully coordinated cross-system atomic transactions that interact with a variety of recoverable resources.

## Prerequisites

This tutorial is written for CICS programmers and administrators whose skills and experience include basic CICS resource management and web services knowledge. You should have a basic familiarity with Unix System Services, XML, SOAP, CICS transactions CEDF, and CEMT.

## System requirements

To run the examples in this tutorial, you need IBM CICS Transaction Server for Z/OS Version 3.1 or greater (see [Resources](#)) with two available CICS regions.

---

## Section 2. The web service

The web service used in this tutorial has been created from `accountControl.wsdl`, which contains two operations: `setBalance` and `adjustBalance`. For the purposes of this example, the balance is stored in a recoverable TSQ with a log of updates.

**Table 1. `accountControl.wsdl` operations**

<code>setBalance</code>	Resets the balance to a given value. If the value is out-of-bounds, sets it to 0.
<code>adjustBalance</code>	Changes the current balance by the amount given in the request. If the adjustment creates an out-of-bounds

```
balance, calls  
SYNCPOINT  
ROLLBACK to cancel  
the transaction.
```

Both the provider and requester update their own recoverable TSQs with the account balance. Using WS-Atomic Transactions, you can keep these synchronized even after a backout on one component in the transaction.

Included with this tutorial is a [downloadable zip file](#) containing the WSDL, bind files, requester application, provider application and pipeline configuration files. The bind and pipeline configuration files should be placed in a directory on HFS, for example, /u/[yourname]/accExample. The three applications, ACCSRV, ACCREQ, and WSATHND, should be placed in a dataset and added to your DFHRPL.

The WSDL file, which describes the service, was used to create the CICS applications and the bind files. It is not required for this tutorial.

This tutorial requires two CICS regions; choose one to use for the web service provider and one for the requester.

## Set up the service

Within your provider CICS region you need to define several resources to set up the pipeline and the web service for use. This section concentrates on the provider (ACCSRV). For the purposes of this tutorial, it is assumed that all service resources are installed in group ACCSRV.

- Define a Tcpservice resource ACCTCP and note it's port number.
- Define a Pipeline resource, ACCSRV, ensuring the fully-qualified path to provider.xml (from the zip file) is the Configfile parameter.
- Define a web service called ACCSRV, using Pipeline ACCSRV (defined above) and the fully-qualified path to accountControlService.wsbind as the WSBind parameter.
- Define a Urimap ACCSRV, with Usage=Pipeline, Host=\*, Path=/accsrv, Pipeline=ACCSRV, Webservice=ACCSRV, Transaction=CPIH.
- Define a Tsmode ACCMODEL with Prefix=ACC+ and Recovery=Yes. This sets up all TSQs starting with prefix ACC to be recoverable.
- If autoinstall is off, you also need to define program ACCSRV as a standard CICS program resource.

Note that three of the resources you have just defined in group ACCSRV are called

ACCSRV. This is perfectly valid in CICS.

Ensure that all resources install correctly and are enabled. The service should now be usable and can be located at URI `http://[host]:[port]/accsrv`, where `host` is the system's hostname and `port` is defined by your `TcpipService` resource.

## Set up the requester

Within your requester CICS you need to define several resources to setup the pipeline for use. This section concentrates on the requester (ACCREQ). For the purposes of this tutorial, it is assumed that all service resources are installed in group ACCREQ.

- Define a `TcpipService` resource, `ACCTCPR`.
- Define a `Pipeline` resource, `ACCREQ`, ensuring the fully-qualified path to `requester.xml` is the `Configfile` parameter.
- Define a web service called `ACCREQ`, using pipeline `ACCREQ` (defined above) and the fully-qualified path to `accountControlClient.wsbind` as the `WSBind` parameter.
- Define Transaction `ACCR` for program `ACCREQ`. Give it a `DTimout` of `0500` and ensure the `Indoubt Action` attribute is `Backout`.
- Define a `Tsmode` `ACCMODEL` with `Prefix=ACC+` and `Recovery=Yes`. This sets up all `TSQs` starting with prefix `ACC` to be recoverable.
- If `autoinstall` is off, you also need to define program `ACCREQ` as a standard CICS program resource.

Ensure that all resources install correctly and are enabled. The client should now be usable.

## Run the web service

As you run a command-line application which requires the use of a URI, lower case characters must be enabled. Type the following to enable the input of lower-case characters on the requester CICS terminal:

```
CEOT TRANID
```

Exit the `CEOT` transaction before continuing.

First set the endpoint your requester will call to the one you have defined in the service set-up section. Type the following:

```
ACCR URI http://[host]:[port]/accsrv
```

using the [host] and [port] used in the ["Set up the service"](#) section. You have to perform this step on each restart, as the URI gets stored in a local TSQ.

As the bank account does not exist initially, you must create it by setting a balance. Type the following to give an initial balance of 100:

```
ACCR SET 100
```

Now adjust the balance using the ADJ operation:

```
ACCR ADJ 24
```

To see what you have just done, you can view the requester application's TSQ and the provider's using CEBR. On the provider system type:

```
CEBR ACCBALANCE
```

On the requester use:

```
CEBR ACCLCLBALANCE
```

Both TSQs should have the initial set of 100, and an adjustment to 124.

### Listing 1. ACCBALANCE and ACCLCLBALANCE TSQ data

```
***** TOP OF QUEUE *****
100
124
**** BOTTOM OF QUEUE ****
```

## Failures

To demonstrate the non-atomic ability of the web service, adjust the balance by -200:

```
ACCR ADJ -200
```

Overdraft features and negative balances are not allowed in this bank so, on reaching the provider, the update is rejected and the service backed-out using the EXEC CICS SYNCPOINT ROLLBACK command.

However, the requester does not check that the update is valid and continues regardless.

The provider TSQ (ACCBALANCE) is unchanged:

### Listing 2. ACCBALANCE (server-side) TSQ data

```
***** TOP OF QUEUE *****
100
124
**** BOTTOM OF QUEUE ****
```

Now, however, the requester's TSQ is inconsistent.

### Listing 3. ACCLCLBALANCE (requester-side) TSQ data

```
***** TOP OF QUEUE *****
100
124
-76
**** BOTTOM OF QUEUE ****
```

Of course there are solutions to this, such as checking for negative balances in both the requester and provider; however, this could lead to inconsistencies in validation code and is not an ideal solution.

In the next step, you enable atomic transactions and attempt to run the same commands again to show that by rolling back the provider, the requester will also be rolled back.

---

## Section 3. A Web Services Atomic Transactions (WS-AT) introduction

Using WS-AT, you can enable web services to ensure atomicity throughout a complete transaction. This means that either the transaction will be completed with all units of work finished, updated, and committed, or the transaction will fail and the state of the recoverable units will return to that of before the transaction began.

Web service Atomic Transactions are described by specifications (listed in the [Resources](#) section) designed to unify the process required for web services to perform transactionally across-application and platform. It involves coordinating a set of messages between endpoints, deciding whether to commit or roll back any web services transaction. The coordination is done between a set of Registration Services.

When the web services requester attempts to invoke a web service under WS-AT, it sends the standard body request along with a set of SOAP header elements initiating the coordination. The complete message flows are as follows:

- Requester invokes web service with initial coordination context. For the case of atomic transactions, the requester is known as the **coordinator**.
- Provider requests to participate by sending a **Register** coordination request. The provider is a **participant** in the transaction.
- Requester sends **RegisterResponse** acknowledging the participant's request.
- Provider executes its application code and responds in the same way as a non-AT web service would.
- All other requester action is taken now. For example, it could call several more atomic web services, update some database records, and so on.
- Once complete, the requester region attempts to initiate completion by sending a **Prepare** message to the provider.
- If in agreement, the provider responds with **Prepared**.
- Requester sends **Commit** to finalize the provider's actions.
- Provider responds with **Committed**. Once these are received for all transactional entities, the requester commits and ends.

If any stage of the process is incomplete or the provider or requester roll back, the transaction will either roll back or wait until further coordination messages are received. This is determined by the specification document.

---

## Section 4. Enable WS-AT in CICS

CICS uses pipelines to act as the Registration Services for the coordination requests. Each web service pipeline must have a requester and provider Registration Service pipeline to send and receive one-way registration requests. They are called `registrationservicePROV.xml` and `registrationserviceREQ.xml` and can be located on HFS under the CICS directory, for example, `/usr/lpp/cicsts/cicsts31/pipeline/configs/`.

### Update the service pipeline configuration file

Edit your HFS-based provider.xml file to look like the following:

#### Listing 4. HFS-based provider.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/http/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.ibm.com/software/http/cics/pipeline
  provider.xsd">
  <service>
    <terminal_handler>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>DFHWSATH</program_name>
        <namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoor</namespace>
        <localname>CoordinationContext</localname>
        <mandatory>false</mandatory>
      </headerprogram>
    </cics_soap_1.1_handler>
  </terminal_handler>
</service>
<apphandler>DFHPITP</apphandler>
<service_parameter_list>
  <registration_service_endpoint>
    http://localhost:[port]/cicswsat/RegistrationService
  </registration_service_endpoint>
</service_parameter_list>
</provider_pipeline>
```

The new header program, DFHWSATH, controls all coordination messages through the registration service pipelines. As you have the mandatory flag set to *false*, atomic transactions will only be coordinated if the ...2004/10/wscoor namespace is included in the incoming message.

The service parameter list contains the endpoint where the registration service pipelines will reside. For this example, install all provider resources on the same CICS region; edit the port number to be the same as ACCTCP, your provider Tcpiptservice.

## Update the requester pipeline configuration file

Edit your requester.xml file to look like the following:

#### Listing 5. HFS-based requester.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline xmlns="http://www.ibm.com/software/http/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
  http://www.ibm.com/software/http/cics/pipeline requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
```

```
<program_name>DFHWSATH</program_name>
<namespace>http://schemas.xmlsoap.org/ws/2004/10/wscoor</namespace>
<localname>CoordinationContext</localname>
<mandatory>true</mandatory>
</headerprogram>
</cics_soap_1.1_handler>
</service_handler_list>
</service>
<service_parameter_list>
  <registration_service_endpoint>
    http://localhost:[port]/cicswsat/RegistrationService
  </registration_service_endpoint>
</service_parameter_list>
</requester_pipeline>
```

Similarly to the provider pipeline, you have added the DFHWSATH supplied atomic transaction application with the same namespace and localname. Mandatory is *true* in this case, as you must force the initial outbound coordination header information.

Update the registration service endpoint as before to point to **ACCTCPR**, your requester's port.

## Install resources

Logged on to CICS, ensure the ACCSRV and ACCREQ pipelines are uninstalled. You must first configure both the provider and requester CICS regions using resources from the supplied DFHWSAT group. The CICS Infocenter contains a sequential guide, copied here for convenience.

1. Copy the contents of group DFHWSAT to another group. All the resources are supplied by CICS in group DFHWSAT, and you cannot change them. However, you will need to change the CONFIGFILE attribute in the PIPELINE resources.
2. Modify the CICS-supplied registration services provider PIPELINE resource. The PIPELINE is named DFHWSATP, and specifies pipeline configuration file /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationservicePROV.xml in the CONFIGFILE attribute.
  - a. Change the CONFIGFILE attribute to reflect the location of the file in your system.
  - b. Leave the other attributes unchanged.

**Important:** Use the pipeline configuration file exactly as provided: do not change its contents.

3. Install the PIPELINE resources. The registration services provider PIPELINE resource need not be in the same CICS region as your service requester or provider applications (for the purposes of this tutorial they are), but must be connected to that region with a suitable MRO connection.

4. Without changing it, install the URIMAP that is used by the registration services provider in the same region as the PIPELINE. The URIMAP is named DFHRSURI.

5. Modify the CICS-supplied registration services requester PIPELINE resource. The PIPELINE is named DFHWSATR, and specifies pipeline configuration file /usr/lpp/cicsts/cicsts31/pipeline/configs/registrationserviceREQ.xml in the CONFIGFILE attribute.

a. Change the CONFIGFILE attribute to reflect the location of the file in your system.

b. Leave the other attributes unchanged.

**Important:** Use the pipeline configuration file exactly as provided: do not change its contents.

6. Install the PIPELINE resources (you require both the provider and requester on each region).

**Important:** The registration services requester PIPELINE resource must be in the same CICS region as the service requester and provider applications.

7. Install the programs used by the registration service provider pipeline in the same region as your PIPELINE resources. The programs are DFHWSATX, DFHWSATR, and DFHPIRS. If both your PIPELINE resources are in different regions, you will need to install these programs in both regions.

8. Install the PROGRAM resource definition for the header handler program. The program is named DFHWSATH. Install the PROGRAM in the regions where your service provider and requester applications run.

You must reinstall pipelines ACCSRV and ACCREQ before the changes to the configuration file are picked up. Disable the two pipelines and discard the resources (using CEMT).

Install pipeline ACCSRV on the provider region and ACCREQ on the requester without modifying any resource attributes. Atomic transactions become activated by the header program DFHWSATH introduced into your ACCREQ and ACCSRV pipelines.

Ensure all resources are installed and enabled.

## Run the complete transaction atomically

Now run the same commands as you did before you enabled WS-AT.

Type the following to enable the inputting of lower case characters on the requester

CICS terminal:

```
CEOT TRANID
```

Exit the CEOT transaction before continuing.

First set the endpoint that you have defined in the service set up section. Type the following:

```
ACCR URI http://[host]:[port]/accsrv
```

using the [host] and [port] used in the "Set up the service" section. You have to perform this step on each restart, as the URI gets stored in a local TSQ.

As the bank account does not exist initially, you must create it by setting a balance. Type the following to give an initial balance of 100:

```
ACCR SET 100
```

Now adjust the balance using the ADJ operation:

```
ACCR ADJ 24
```

To see what you have just done you can view the requester application's TSQ and the provider's using CEBR. On the provider system type:

```
CEBR ACCBALANCE
```

On the requester use:

```
CEBR ACCLCLBALANCE
```

As before, both TSQs have the initial balance set to 100, and an adjustment to 124.

### Listing 6. ACCBALANCE and ACCLCLBALANCE TSQ data

```
***** TOP OF QUEUE *****
100
124
**** BOTTOM OF QUEUE ****
```

## Failures

To demonstrate the atomic nature of the cross-system transaction, attempt the same failure case you tried earlier, which shows that both TSQs stay synchronized.

Adjust the balance by -200:

ACCR ADJ -200

The provider rejects the adjustment because it results in a negative balance. It issues EXEC CICS SYNCPOINT ROLLBACK as before. This time the requester's recoverable actions are rolled back so both TSQs look the same:

### Listing 7. ACCBALANCE and ACCLCLBALANCE TSQ data

```
***** TOP OF QUEUE *****
100
124
**** BOTTOM OF QUEUE ****
```

In fact, the requester application appends, with code ASP7, "A resource manager involved in syncpoint protocols has replied 'No' to a request to 'Prepare.'"

---

## Section 5. Extension: View coordination messages

While running a web service using WS-AT, there are a set of coordination messages that flow through the registration service Pipelines. These are for Register, RegisterResponse, Prepare, Prepared, Commit, and Committed. Using the program WSATHND, you can trace these message flows in CEEOUT. Note that you can see these messages using auxtrace, but for this tutorial I can demonstrate more clearly using the program.

### Set up the registration services to use WSATHND

WSATHND is an application program that sits on the two Registration Service pipelines and logs the current coordination action being flowed. You will use it for the purpose of this tutorial to demonstrate the coordination message flow between requester and provider.

Create a copy of the registrationservicePROV.xml configuration file and modify it to look like the following:

### Listing 8. Modification of registrationservicePROV.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<provider_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
```

```

"http://www.ibm.com/software/htp/cics/pipeline_provider.xsd">
<service>
  <service_handler_list>
    <cics_soap_1.1_handler>
      <headerprogram>
        <program_name>WSATHND</program_name>
        <namespace>*</namespace>
        <localname>wsatHeader</localname>
        <mandatory>>true</mandatory>
      </headerprogram>
    </cics_soap_1.1_handler>
  </service_handler_list>
  <terminal_handler>
    <handler>
      <program>DFHWSATX</program>
      <handler_parameter_list/>
    </handler>
  </terminal_handler>
</service>
<service_parameter_list/>
</provider_pipeline>

```

Similarly with registrationserviceREQ.xml:

### Listing 9. Modification of registrationserviceREQ.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<requester_pipeline
  xmlns="http://www.ibm.com/software/htp/cics/pipeline"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation=
    "http://www.ibm.com/software/htp/cics/pipeline_requester.xsd">
  <service>
    <service_handler_list>
      <cics_soap_1.1_handler>
        <headerprogram>
          <program_name>WSATHND</program_name>
          <namespace>*</namespace>
          <localname>wsatHeader</localname>
          <mandatory>>true</mandatory>
        </headerprogram>
      </cics_soap_1.1_handler>
    </service_handler_list>
  </service>
</requester_pipeline>

```

You have added WSATHND to be run as a header handler every time a Registration Service is invoked.

Now update the resource definitions in CICS to find these new registration service configuration files. Do the following on both the requester and provider CICS regions:

- Disable DFHWSATP and DFHWSATR pipelines.
- Discard DFHWSATP and DFHWSATR pipelines.
- Alter the resource definition for pipeline DFHWSATP by pointing the

Configfile attribute to your modified version of registrationservicePROV.xml.

- Alter the resource definition for pipeline DFHWSATR by pointing the Configfile attribute to your modified version of registrationserviceREQ.xml.
- Reinstall the two pipelines (DFHWSATR, DFHWSATP).

## Run an atomic transaction and view messages

As you did previously, you can try invoking some web services atomically.

As you already have bank account information in the two TSQs, continue from the point previously. Your TSQs should look like the following:

### Listing 10. Current ACCBALANCE and ACCLCLBALANCE state

```
***** TOP OF QUEUE *****
100
124
**** BOTTOM OF QUEUE ****
```

First attempt to adjust the balance successfully:

```
ACCR ADJ -20
```

Take a look at the CICS logs, and in particular CEEOUT. The requester contains the following:

### Listing 11. Requester CEEOUT log output

```
WSAT: REACHED HANDLER - RECEIVE-REQUEST
WSAT: ACTION:Register      :
WSAT: REACHED HANDLER - SEND-REQUEST
WSAT: ACTION:RegisterResponse:
WSAT: REACHED HANDLER - SEND-REQUEST
WSAT: ACTION:Prepare      :
WSAT: REACHED HANDLER - RECEIVE-REQUEST
WSAT: ACTION:Prepared     :
WSAT: REACHED HANDLER - SEND-REQUEST
WSAT: ACTION:Commit      :
WSAT: REACHED HANDLER - RECEIVE-REQUEST
WSAT: ACTION:Committed   :
```

The provider contains the following:

### Listing 12. Provider CEEOUT log output

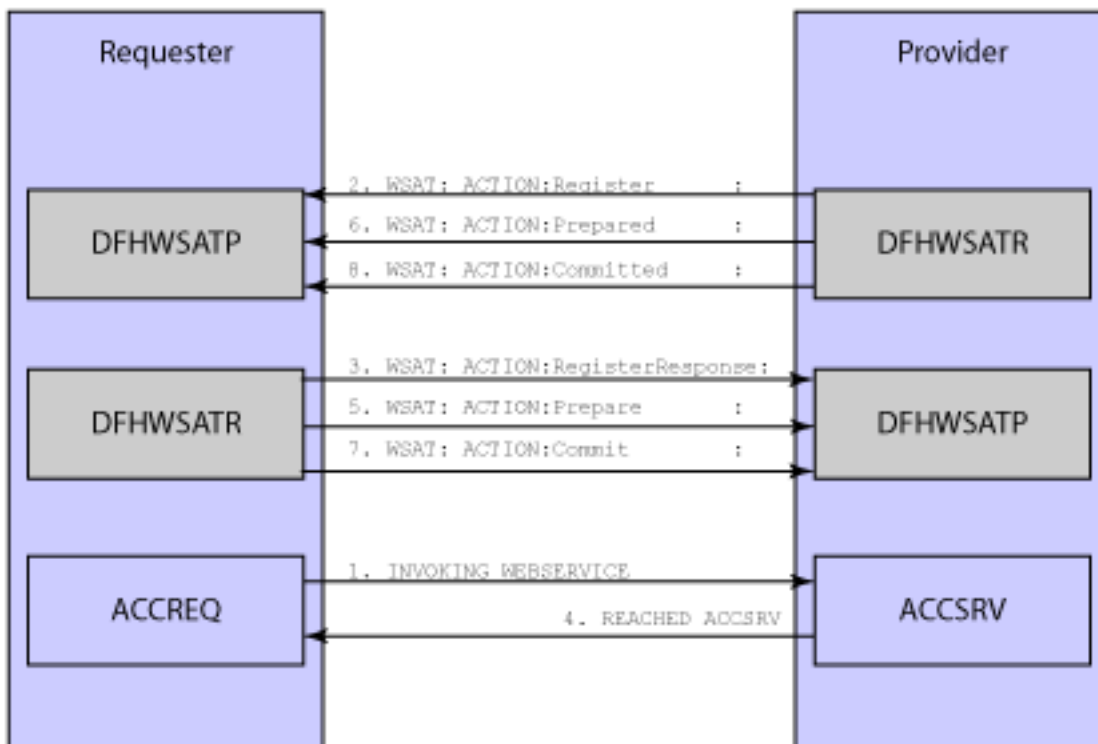
```
WSAT: REACHED HANDLER - SEND-REQUEST
WSAT: ACTION:Register      :
```

```

WSAT: REACHED HANDLER - RECEIVE-REQUEST
WSAT: ACTION:RegisterResponse:
REACHED ACCSRV
WSAT: REACHED HANDLER - RECEIVE-REQUEST
WSAT: ACTION:Prepare :
WSAT: REACHED HANDLER - SEND-REQUEST
WSAT: ACTION:Prepared :
WSAT: REACHED HANDLER - RECEIVE-REQUEST
WSAT: ACTION:Commit :
WSAT: REACHED HANDLER - SEND-REQUEST
WSAT: ACTION:Committed :
    
```

You can see a whole host of messages between the requester and provider. [Figure 1](#) shows the order in which they are sent.

**Figure 1. WS-AT commit coordination messages**

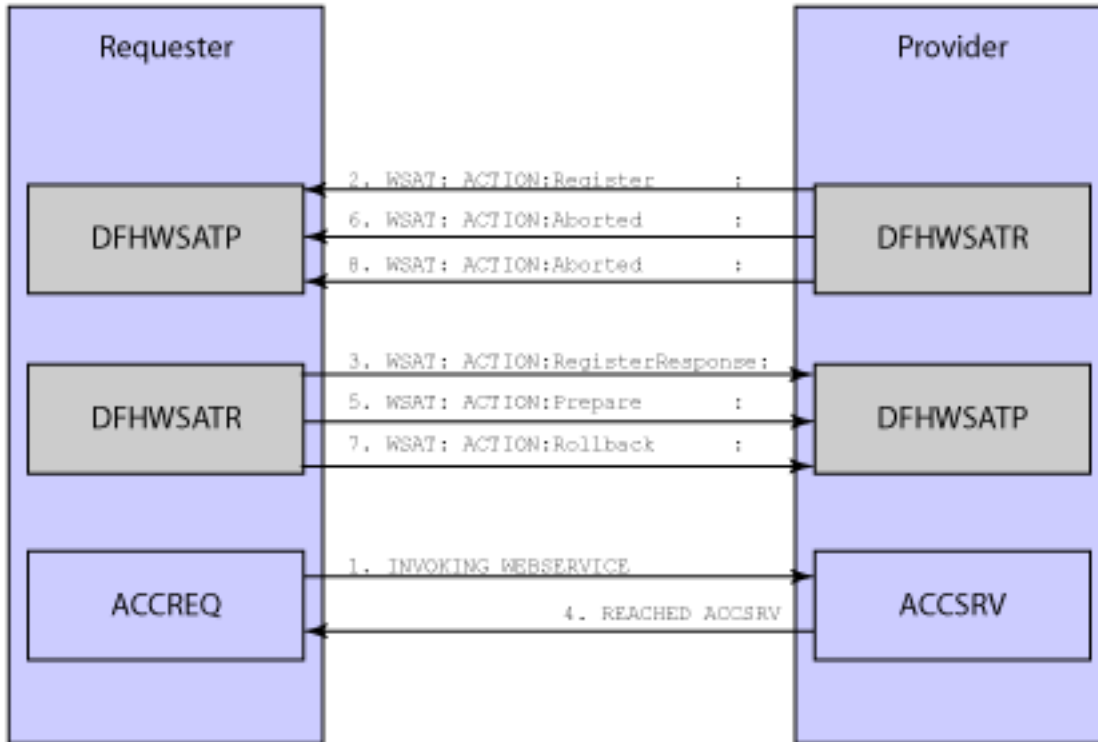


Now run the following comand to give a failure case:

```
ACCR ADJ -250
```

The logs now show the following message flows, leading to a rolled-back transaction initiated by the provider recovery manager.

**Figure 2. WS-AT roll back coordination messages**



So far you have invoked a normal web service using a requester and provider both within CICS regions. You then enabled WS-AT and, using a recoverable TSQ, showed that simple changes can be backed out throughout the transaction. In the next sections, you can explore the possibilities for adding more complex resources and interacting with IBM WebSphere® Application Server's web services atomic transactions.

## Section 6. Next steps

You have seen the basic scenario using CICS as a requester and provider to show the effects of turning WS-AT on and to see message flows between the client and the endpoint. A more realistic, business-oriented application could call multiple web services, update more complex resources, and interact with non-CICS web services gateways.

### A more complex scenario

The example you have run through in this tutorial is a simple one; you are only calling a single web service from a simple application program. This web service

updates a recoverable TSQ and can roll back if necessary. More realistically, the recoverable resource would be more complex, such as a VSAM file or DB2® UDB database record.

Using WS-AT does not affect the way in which individual CICS applications can perform transactionally except to provide the additional, cross-system functionality we require. All in-application calls made to DB2 UDB, writing to VSAM files, or any other recoverable task can be dealt with in conjunction with atomic web services by the CICS recovery manager. All tasks must complete successfully for the changes to be committed as expected.

You can also use multiple atomic web service calls from within a single transaction. The requesting application will successfully call several web services and coordinate them all to completion. Similarly the providing application can successfully call several web services and act as both a participant for its requester and coordinator as a requester for its invoked web services.

## WebSphere enablement

You can use the WSDL document from your web service to create a WebSphere Application Server (Application Server) skeleton web service, which you can use to interact with CICS. IBM Rational® Application Developer can import WSDL files and convert them into a web services provider or requester. Atomic transactions support is available and can be implemented for this process. These transactions can then be exported to Application Server (Version 6.02 minimum). The details of how this is done is beyond the scope of this tutorial, however once enabled and running, your CICS set-up need not change to call the WebSphere provider.

Simply change the endpoint URI from your requester application (**ACCR URI http://.../accsrv**) to call a new endpoint. The WSDL document and specifications define the contents of the message flows, meaning they are standardized and will work independently of the software used to create the application code.

Direct Application Server's web services requester at your CICS endpoint (**http://[host]:[port]/accsrv**) passing the parameters defined in the WSDL (operation SET or GET with a value). Once enabled, web services atomic transaction coordination works between Application Server and CICS, with message flow and ordering being identical to what you have seen.

## Downloads

Description	Name	Size	Download method
Resources for this tutorial	ws-atxincicscode.zip	31KB	<a href="#">HTTP</a>

[Information about download methods](#)

# Resources

## Learn

- [Web Services Transactions Specifications](#) -- includes WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity.
- [CICS TS for z/OS V3.1 Infocenter](#).
- [WebSphere Application Server V6 Infocenter](#).
- [Standards roadmap](#) -- understand the impact and importance of standards and specifications for the development of SOA and web services.
- [SOA and web services](#) -- hosts hundreds of informative articles and introductory, intermediate, and advanced tutorials on how to develop web services applications.

## Get products and technologies

- Download trial versions of the following products:
  - [WebSphere Application Server](#)
  - [Rational Application Developer](#)
  - [DB2 Universal Database Express Edition](#)
- [IBM CICS Transaction Server for Z/OS](#) -- find more information.

## Discuss

- [developerWorks blogs](#) -- get involved in the developerWorks community.

## About the author

Oliver R. Fenton

Oliver Fenton worked in the CICS Test Department in Hursley Park, UK during the development of CICS Transaction Server Version 3.1 where he worked primarily on the web services feature, focusing especially on Atomic Transactions. He has experience with web services in CICS, .NET, and WebSphere Application Server and has recently moved to a new role servicing Java technology in the Java Technology Center at Hursley Park.