Understand how the new features
in CICS improve Java workloads
and application lifecycle management

August 2011

Running Java workloads with JVM servers and OSGi

Fraser Bohm, Lead Developer, CICS Transaction Server
Paul Cooper, Developer, CICS Transaction Server
Ben Cox, Developer, CICS Transaction Server
Elisabetta Flamini, Developer, CICS Transaction Server
Ivan Hargreaves, Developer, CICS Transaction Server
Matthew Webster, Technical Strategist, CICS Transaction Server

CICS icon of progress: http://www.ibm.com/ibm100/us/en/icons/cics/

Executive Summary

This paper sets out the compelling enhancements to the Java support in CICS® Transaction Server Version 4.2. Although it has been possible to run Java applications in CICS Transaction Server (CICS TS) since Version 1.3, the significant strides made in CICS TS 4.2 represent a game-changing shift in the technology.

This paper describes a number of areas where both significant evolutionary enhancements, as well as fundamental architectural changes, have dramatically raised the value proposition for having CICS applications written in Java and for reusing existing Java components to the benefit of existing and new CICS applications. In particular the paper explores the benefits of the JVM server hosting capabilities, the new underlying 64-bit JVM, OSGi support for application development and deployment, and the use of Axis2 for Java-based web services.

These capabilities provide the following key benefits:

- The JVMSERVER resource provides users with a single resource to manage a complete JVM and related application hosting capabilities. It also radically reduces the storage required to satisfy concurrent tasks. Each task will run on a thread in the JVM, rather than requiring a whole JVM.
- Concurrency is vastly increased, with up to 256 transactions running simultaneously in the same JVM server. The reduced overheads, along with generous heap sizes available in 64-bit JVMs, offer significantly better Java scalability than in previous versions.
- The JVM server represents a move to a more industry standard Java experience in CICS. This results in a number of distinct benefits:
    - The JVM server provides a more industry standard server-side programming model. Applications can share an 'engine' as well as sharing data between tasks. This greatly improves the potential for reuse of existing Java components.
    - Support for the OSGi standard means that applications written for the JVM server (and migrated from pooled JVMs) will be packaged as OSGi bundles. This allows dynamic deployment, replacement, and versioning of applications within the JVM server. In other words, you can upgrade, enhance, or patch applications without restarting the JVM.
    - Standard tools can be used for debugging, problem determination, and performance monitoring. This allows for skills reuse and knowledge transfer.
- CICS Explorer™ provides development support, allowing applications to be developed on the client PC in familiar tools such as Eclipse and Rational® Application Developer for System z®. This allows significant reuse of development skills and knowledge.
- Costs are reduced through the use of IBM® System z Application Assist Processors (zAAP) and IBM System z Integrated Information Processors (zIIP).
- Rapid web service development is provided in Java through support for JAX-WS applications.

Introducing the JVM Server

CICS support for Java in CICS TS 4.2 has taken a huge leap forward. A new model for Java has been introduced which provides a wealth of benefits over previous releases. This new model offers vast storage savings, massively increased concurrency, greatly increased scalability - and a familiar and standardized Java development environment.

While these are just some of the headline benefits, further benefits are realized by the model's stance as a first class CICS resource. The JVMSERVER resource acts as a focal point for configuration, provides centralized control, and manages the interface between CICS and the underlying JVM.

The JVM server

The JVM server model is essentially a single, long-running JVM placed under the control of CICS. Its defining principle is the ability to run many CICS tasks concurrently within the same JVM. As one might expect, compared with the pooled JVM model which requires an exclusive JVM per task, the storage savings are vast.

In the new model, each CICS task that requires Java support is switched onto a thread within the JVM. The thread is taken under the control of CICS dispatcher and becomes a new CICS OTE TCB type called a T8.

Not only can multiple CICS transactions target the same JVM and run concurrently, but every CICS task in this environment is capable of executing Java code, and perhaps more importantly, of making CICS calls and accessing CICS resources and data from Java.

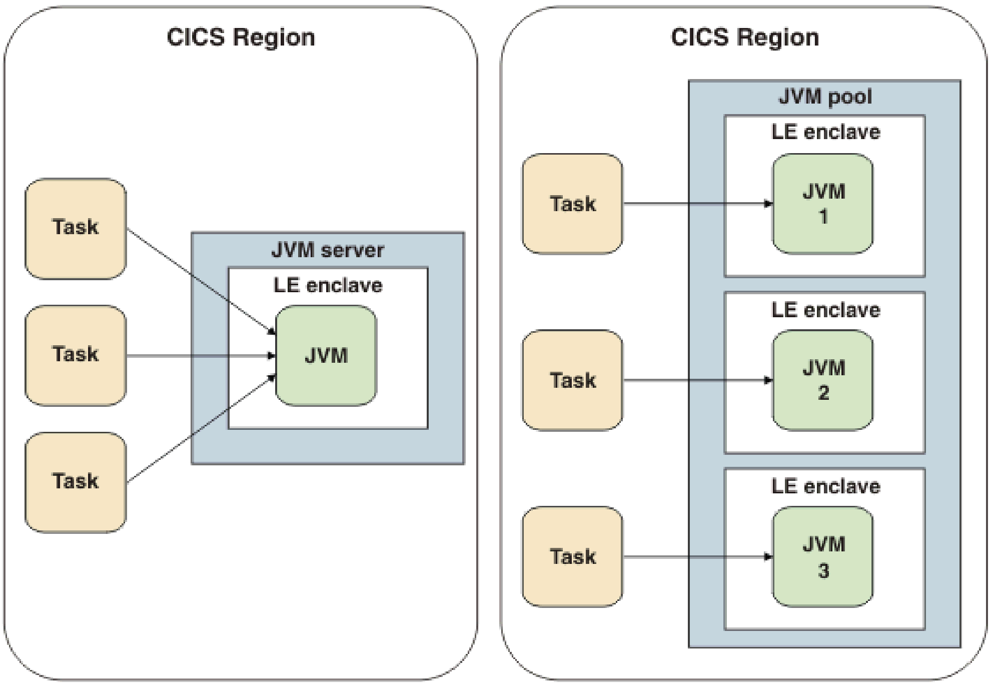


Figure 1. Comparison of a JVM server and pooled JVM environment

The JVMSERVER resource

The foundation of this new model is the JVMSERVER resource. The JVMSERVER resource encapsulates the JVM and provides control over its life cycle. By keeping the JVM lifecycle distinct from the run time, start-up costs and application bootstrap times are no longer incurred on the critical path.

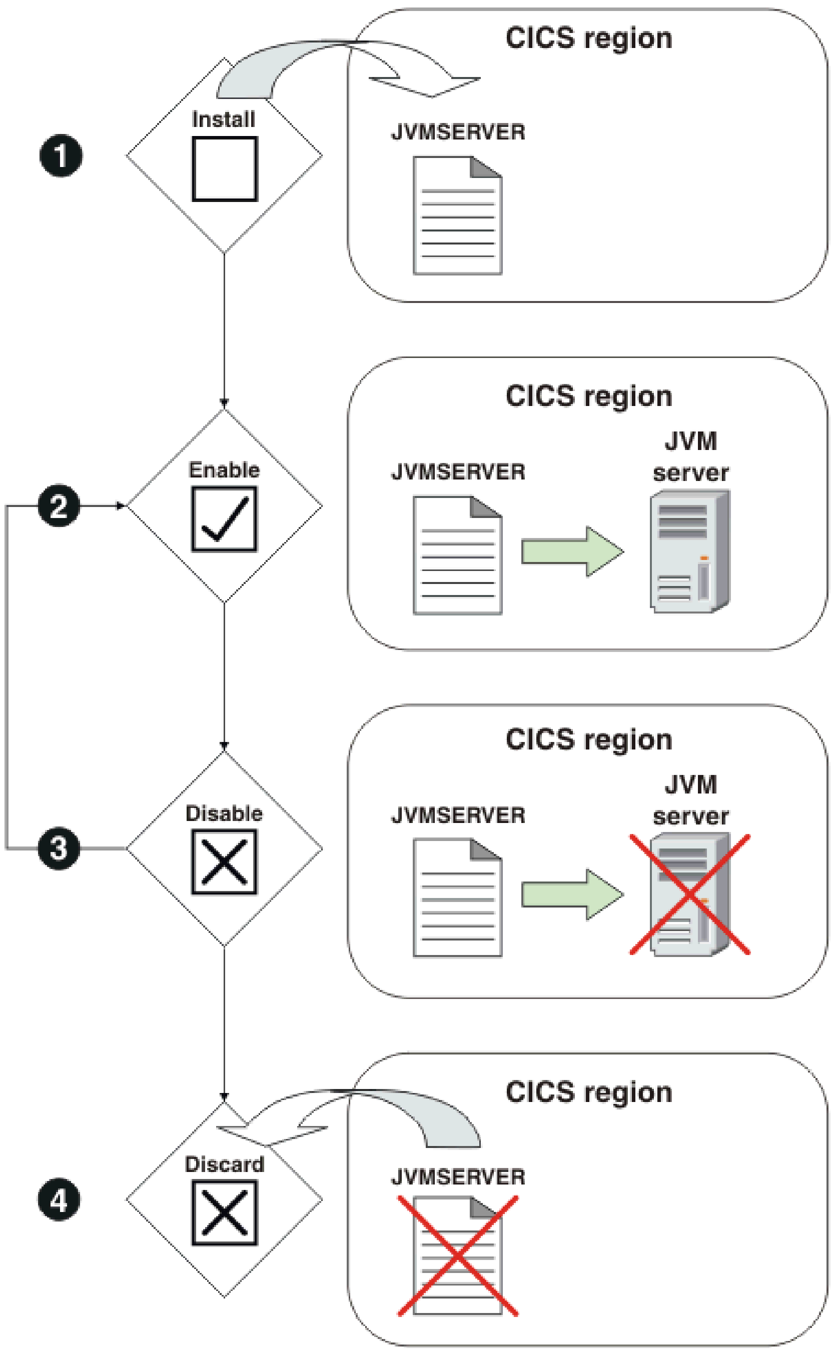


Figure 2. Life cycle of JVMSERVER resource and JVM server

The JVMSERVER resource also provides a configurable thread limit. The thread limit determines the number of threads that a JVM server can run concurrently (and therefore the number of tasks it can serve concurrently). By varying this value the customer can throttle the number of threads (T8 TCBs) available for concurrent activity. Any request that arrives while all threads are in action will cause that task to queue until a thread becomes available.

Additional fine tuning and low-level configuration of both the JVM and the underlying Language Environment enclave can be achieved using the Language Environment runtime options and JVM profile artifacts. Each time the JVMSERVER resource is enabled, it rereads configuration from these files, allowing customers to change the Language Environment enclave characteristics or the JVM settings. Typically customers may want to generate Language Environment storage reports, or tune the JVM heap size, garbage collection model, or JIT settings.

The resource also provides a convenient way to view the workload and other metrics. Information is displayed for values such as the thread count, maximum and current heap sizes, heap occupancy, cumulative thread count, and garbage collection.

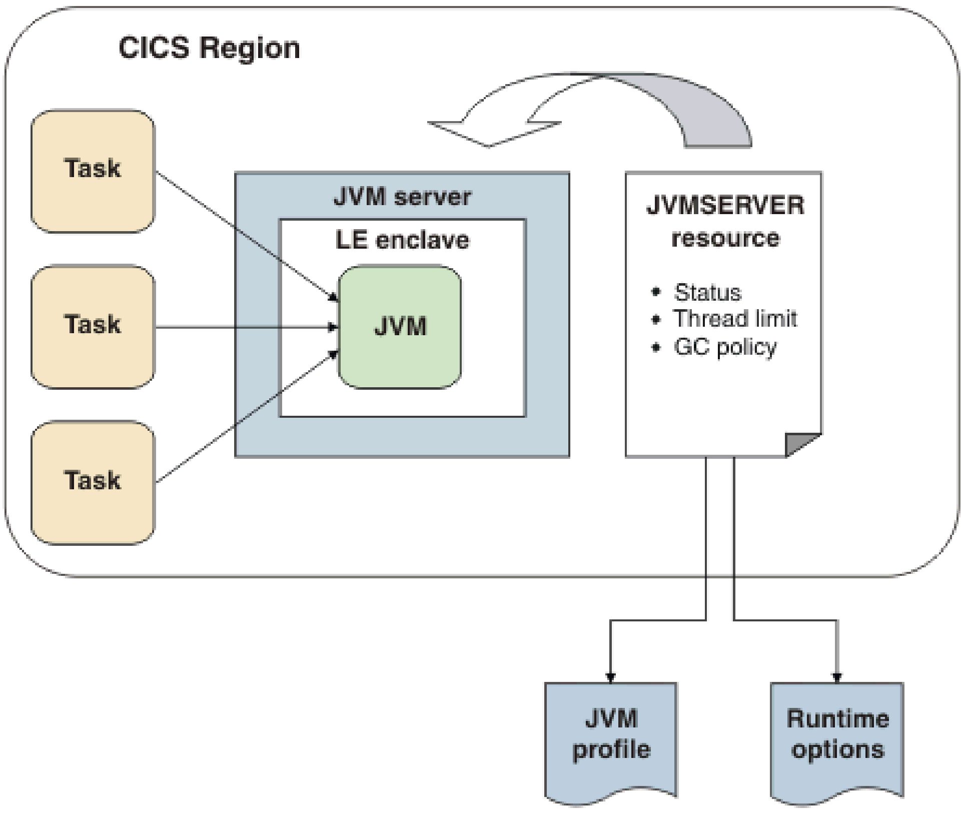


Figure 3. JVMSERVER resource provides configuration for JVM server on start up

Scalability, concurrency, and performance

The JVM server gives CICS the ability to handle many more Java tasks in one region than ever before. Impressively, the JVM server also scales horizontally – this means that multiple JVM servers can be deployed into a single CICS region. Theoretically you could even choose to deploy hundreds of JVM servers into one CICS region.

The beauty of this horizontal scaling is two-fold. Firstly, application and workload isolation can be achieved, and secondly, perhaps more importantly, each JVM server can be configured with a different set of runtime components or even different sets of middleware components. This flexibility keeps each server lightweight and specific to a particular need.

Note that each JVM server has a maximum thread limit of 256, and each CICS region has a limit of 1024 T8 TCBs over all JVM servers. Furthermore, the JVM server utilizes a 64-bit JVM. Heap sizes are no longer constrained by 31-bit storage, and the number of JVMs you can run in the same CICS address space is significantly increased.

Despite the added complexity of thread management, TCB dubbing, and the JNI attach of native threads, the CPU path length of JVM server tasks compared to pooled JVMs in CICS TS 4.2 are extremely closely matched. In effect, by using a JVM server you can achieve big storage reductions and enhanced scalability for no extra CPU cost.

CICS TS 4.2 uses the IBM 6.0.1 64-bit JVM (Java Virtual Machine) instead of the 31-bit JVM. In a 64-bit Java environment, the Java heap is moved above the bar, alleviating 31-bit storage constraints. This allows for greater scalability of applications; Java heap sizes can be increased to cope with increasingly complex applications and to allow highly concurrent workloads to be run in a JVM server. It also allows for CICS region consolidation, with the capability to run many more JVMs in a single CICS address space. The IBM 6.0.1 JVM is additionally optimized for the zEnterprise® hardware, offering significant performance improvements over previous IBM JVMs.

All of these improvements to the Java model lead to scalability way beyond that of previous releases of CICS. The JVM server, with its reduced overheads and large 'above the bar' heap, has yielded significant advantages for CICS TS 4.2.

Server-side Java enablement

The benefits, however, don't stop there. With the advent of JVM server comes a significant increase in the types of Java workload that can run in CICS. Most notable is the ability to run a 'server' application within a JVM server and handle multiple requests for service. Due to its (now) more standard architecture, CICS TS 4.2 has been opened up to countless Java frameworks – it gives CICS the potential to plug in 'application servers' as and when new capabilities become available. A further benefit of using a single shared JVM, and particularly relevant to the 'application server' model, is the ability to share data and state.

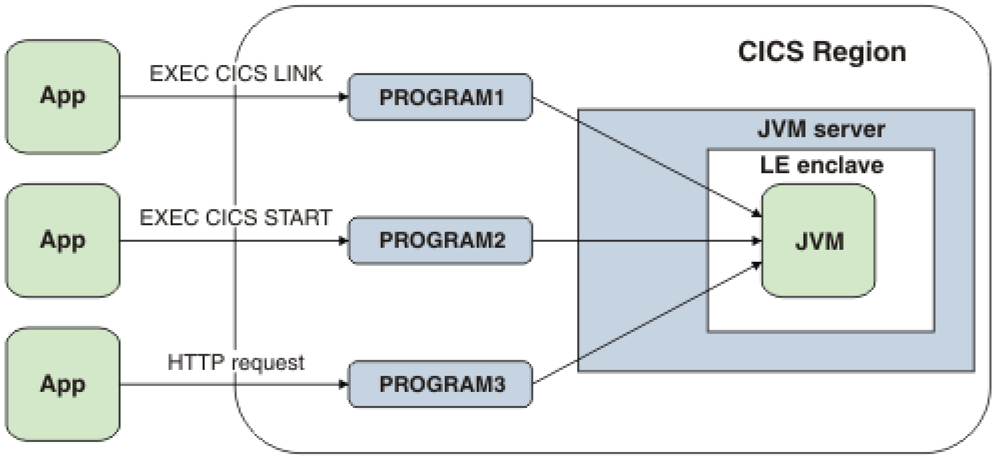


Figure 4. Routing options for running work in a JVM server

Application development and deployment

Traditionally it has been very difficult to develop and deploy mainframe applications from a PC based environment. However, with strong tooling support from the CICS Explorer SDK it is no longer an inhibitor; CICS now has a modern and flexible development and deployment offering that enriches the end-to-end Java experience.

The CICS Explorer SDK combines all the power of Eclipse and its Java development kit, with a deployment model that allows you to create and upload application code directly into the JVM server.

This ease of deployment can be achieved because the JVM server runs an OSGi framework into which applications can be deployed. OSGi is a component model for Java, offering very attractive benefits such as dynamic versioning of application code and dependency checking. It also removes the reliance on error-prone class paths. Essentially, you can upgrade, enhance, version, or patch applications without restarting the JVM.

As you would expect, the Explorer SDK comes with all the CICS SM perspective abilities of its lightweight ‘Explorer’ counterpart. You don’t need to leave the comfort of Eclipse for all your resource definitions and installation work. Moreover, CICS Explorer is the recommended way to view your OSGi bundles and OSGi services (applications).

Finally, there is the added benefit of standard Java debugging. Eclipse can attach a debugger straight to the JVM server. The single JVM makes it significantly easier to debug than the pooled model because all workloads appear as threads in the JVM - previously even determining which JVM you needed to attach to was challenging.

OSGi and CICS

OSGi in CICS complements CICS support for service-oriented architecture (SOA) applications. Web services and Software Component Architecture (SCA) enables CICS as a platform to run SOA enterprise applications and integrates CICS in a large SOA enterprise.

OSGi support allows you to manage and run Java applications based on the main principles that inspired SOA: service based, reusable, and loosely coupled.

OSGi places itself between the service-oriented architectures and the object-oriented systems. While SOA enterprises are constituted of large integrated systems and object-oriented systems are composed of collaborative objects, OSGi is based on components (*bundles*) that are coarser than objects, but with all of them residing in one system.

Modularity and use of services in OSGi allow Java applications to decouple data from the business logic and to dynamically discover and bind new services registered at run time in an OSGi service registry.

Services in OSGi also have the merit to solve one of the main limitations of object-oriented programming: decoupling instances of objects (otherwise bound at instantiation time). Because OSGi is component oriented, services provided by bundles can be reused and shared among different Java applications.

OSGi provides a set of Java Application Programming Interfaces (APIs) to handle the life cycle and the versioning of bundles. Bundles can be dynamically installed and uninstalled and their exposed services registered or made unavailable. Multiple bundle versions can coexist in the same OSGi framework and bundles upgrade can be transparent to the user applications, as continuity of services can be granted during the upgrade.

How CICS exploits OSGi

CICS uses the Equinox version 3.6.1 implementation of the OSGi framework, which supports version 4 of the OSGi Service Platform specification. Through OSGi, CICS implements the dynamic deployment of Java applications into the JVM, as opposed to the traditional class path method, which requires new application classes to be added to the class path, and the JVM restarted to install or update Java applications.

BUNDLES are the resources used by CICS to control the life cycle (install/uninstall/start/stop/register services/unregister services) of OSGi bundles. The new CICS Explorer SDK allows you to specify CICS main classes into the OSGi bundle manifest file. When an OSGi bundle is deployed into a JVM server, CICS registers the specified main classes into the OSGi service registry, and the new services become dynamically available to be used by other bundles in the framework or by CICS when a CICS Java program is linked.

The following figure illustrates how CICS binds an OSGi service when a CICS Java program is run.

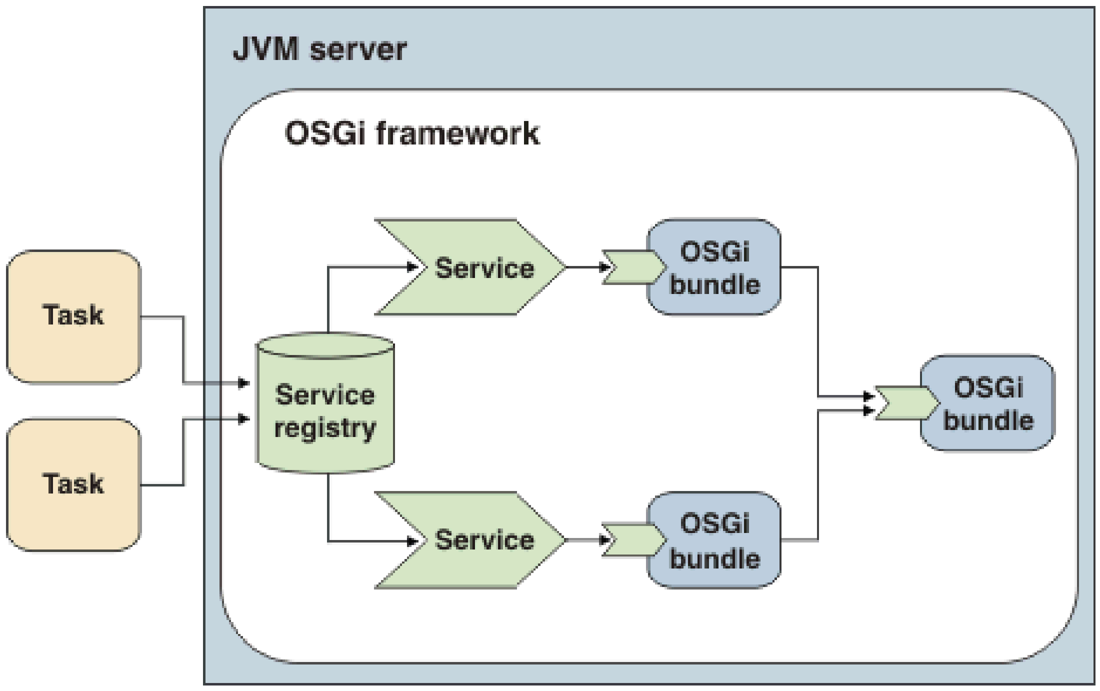


Figure 5. When a Java program is called, the corresponding OSGi bundle is looked up in the service registry and bound

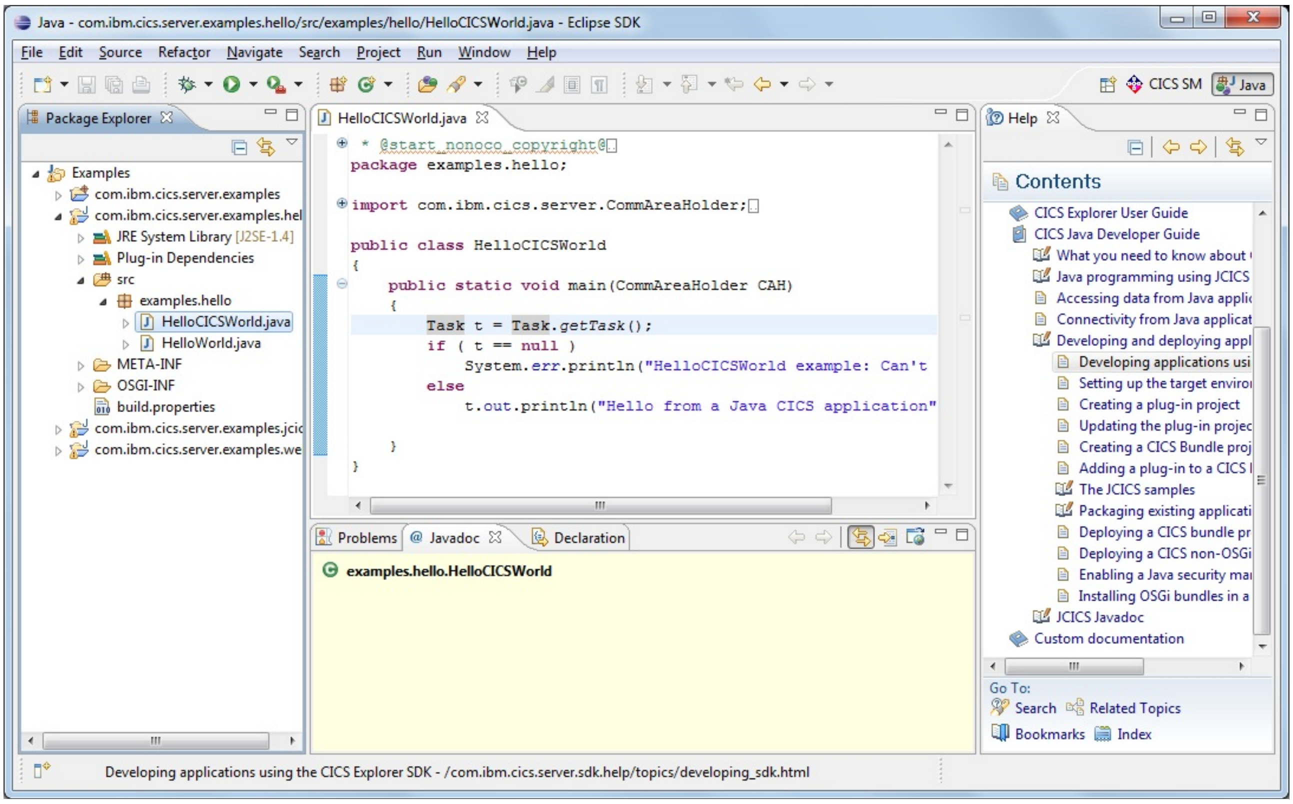
CICS relies on OSGi to provide a smooth and continuous Java application upgrading path. Duplicate disabled services, installed by multiple versions of the same bundle, automatically become active when the currently active service is unregistered.

The CICS Explorer SDK provides the ability to specify aliases for duplicate services, so that multiple versions of the same bundle can be simultaneously operational on the same JVM server.

OSGi support in CICS TS 4.2 makes the JVM server a flexible and maintainable platform for the deployment and the execution of available and reliable Java applications.

CICS Explorer SDK

The CICS Explorer SDK plug-in delivers an end-to-end experience for developing, deploying, and managing CICS Java applications. When the plug-in is installed into the appropriate prerequisite Eclipse-based IDE, users can take advantage of the concise documentation, built-in examples, and simple deployment to have a CICS application running in minutes. Furthermore the full-function CICS Explorer perspectives allow those with an administrator role to create and install the necessary JVM server or pooled JVM environment and CICS resource definitions.


Java - com.ibm.cics.server.examples.hello/src/examples/hello/HelloCICSWorld.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help
CICS SM
Java
Package Explorer
Examples
com.ibm.cics.server.examples
com.ibm.cics.server.examples.hel
JRE System Library [J2SE-1.4]
Plug-in Dependencies
src
examples.hello
HelloCICSWorld.java
HelloWorld.java
META-INF
OSGI-INF
build.properties
com.ibm.cics.server.examples.jcic
com.ibm.cics.server.examples.we
HelloCICSWorld.java
Problems
Javadoc
Declaration
examples.hello.HelloCICSWorld
Help
Contents
CICS Explorer User Guide
CICS Java Developer Guide
What you need to know about
Java programming using JCICS
Accessing data from Java applic
Connectivity from Java applicat
Developing and deploying appl
Developing applications usi
Setting up the target enviro
Creating a plug-in project
Updating the plug-in projec
Creating a CICS Bundle proj
Adding a plug-in to a CICS
The JCICS samples
Packaging existing applicati
Deploying a CICS bundle pr
Deploying a CICS non-OSGi
Enabling a Java security mai
Installing OSGi bundles in a
JCICS Javadoc
Custom documentation
Go To:
Search
Related Topics
Bookmarks
Index

```
 * @start_nonoco_copyright@
package examples.hello;

import com.ibm.cics.server.CommAreaHolder;

public class HelloCICSWorld
{
    public static void main(CommAreaHolder CAH)
    {
        Task t = Task.getTask();
        if ( t == null )
            System.err.println("HelloCICSWorld example: Can't
        else
            t.out.println("Hello from a Java CICS application"

    }
}
```

Developing applications using the CICS Explorer SDK - /com.ibm.cics.server.sdk.help/topics/developing_sdk.html

Figure 6. Screen capture of the CICS Explorer SDK

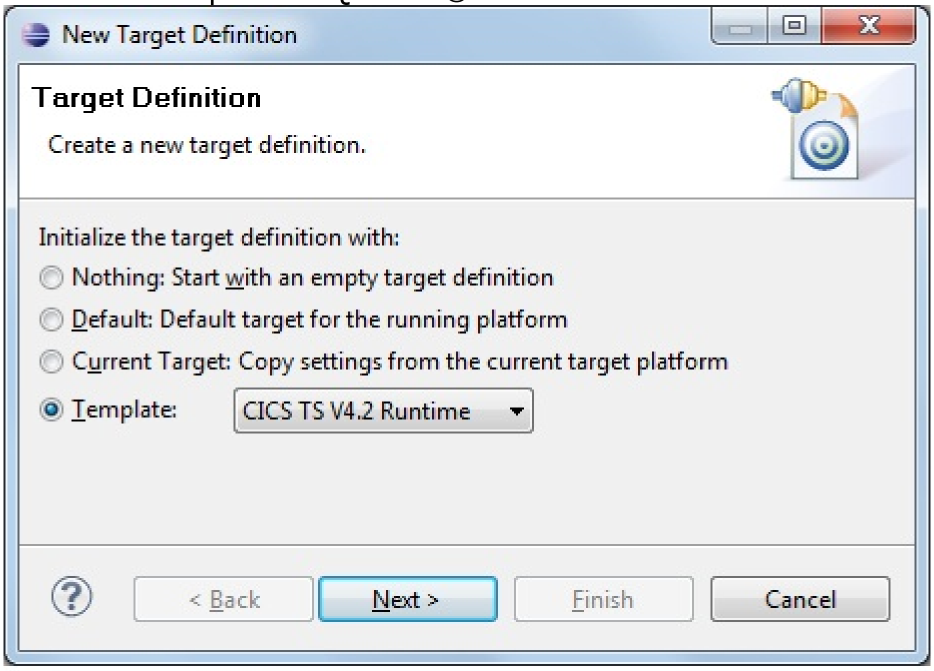
An application that runs in a JVM server is developed as one or more Eclipse plug-in projects, each of which produces an OSGi bundle. An accompanying CICS bundle project is used to package, deploy, and install the finished application into a particular

CICS region or CICSplex, utilizing exactly the same wizards and process as event processing and Atom feeds. This allows different combinations of code and configuration, for example a JVM server name, to be used in certain environments. It also allows platform independent code or libraries to be shared between CICS and other OSGi-enabled server runtime environments such as WebSphere® Application Server.

There are 5 easy steps to getting a CICS Java application running:

1. Define your target platform.

   Select the **CICS TS 4.2 Runtime** target to use the latest Version 4.2 JCICS and Java 6. This also ensures you don't accidentally use an API that is not available in the intended CICS runtime environment. Optionally customize the target by adding any IBM, locally written, or third party library dependencies you need such as WebSphere MQSeries®.



New Target Definition

Target Definition

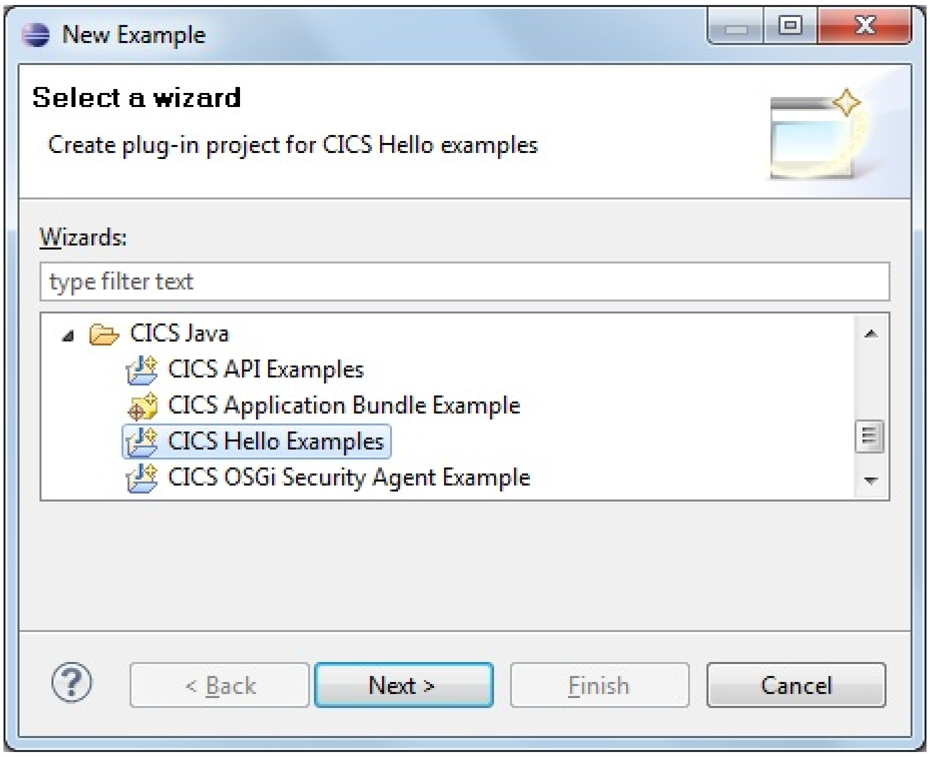
Create a new target definition.

Initialize the target definition with:

- Nothing: Start with an empty target definition
- Default: Default target for the running platform
- Current Target: Copy settings from the current target platform
- [✓ Template:] CICS TS V4.2 Runtime

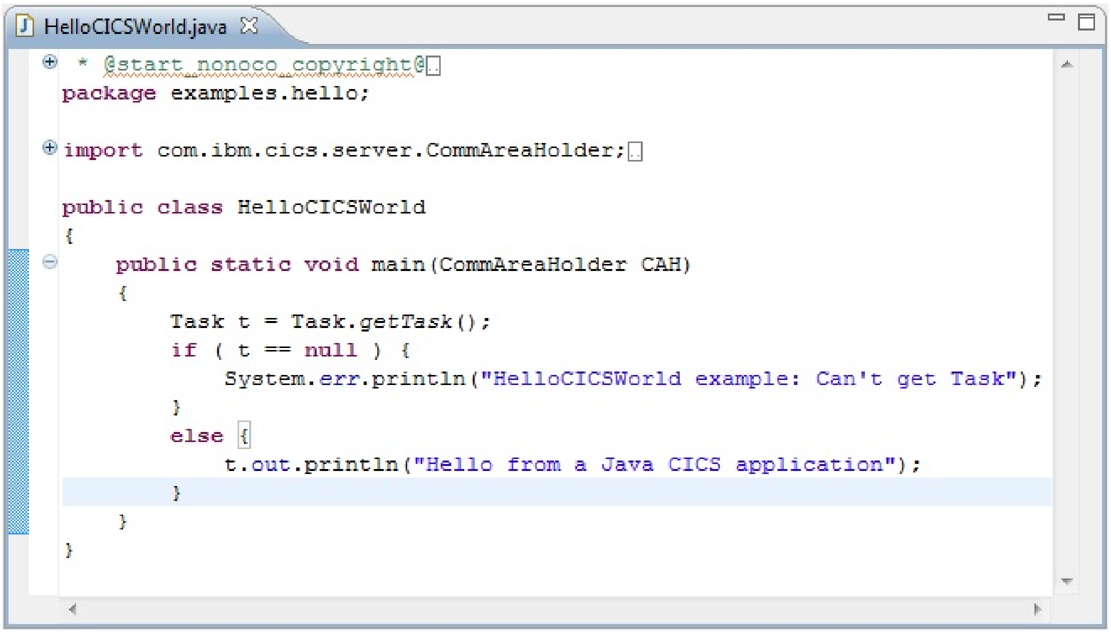
2. Create your Eclipse plug-in project.

   Use the standard New Project wizard to create your own OSGi bundle, or choose one of the samples provided. Add any required packages to the OSGi bundle manifest. Your previous choice of target platform ensures only packages in your CICS server environment are available.



3. Write your CICS program.

   Create a standard CICS main class POJO as with previous versions of CICS. Direct access to Javadoc for all JCICS APIs is available from within the SDK. Declare the new class in the OSGi manifest so that CICS can find it at run time.



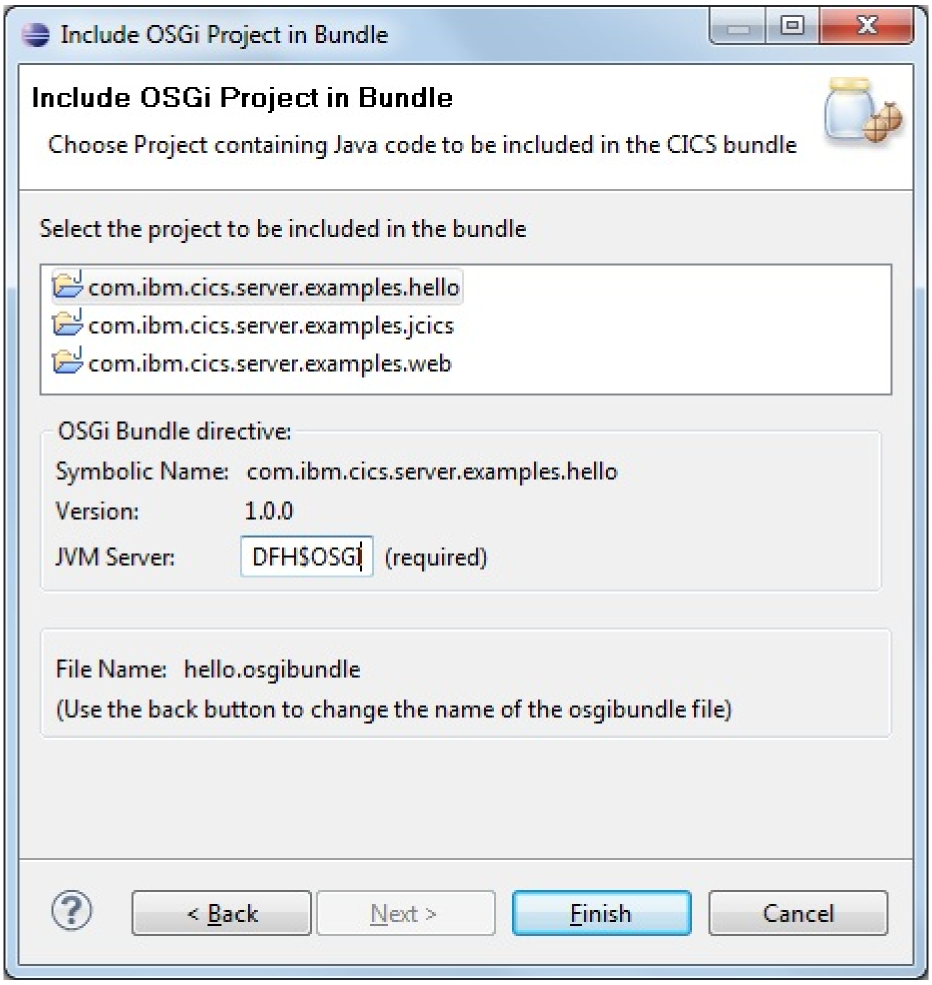
```
 * @start_nonoco_copyright@
package examples.hello;

import com.ibm.cics.server.CommAreaHolder;

public class HelloCICSWorld
{
    public static void main(CommAreaHolder CAH)
    {
        Task t = Task.getTask();
        if ( t == null ) {
            System.err.println("HelloCICSWorld example: Can't get Task");
        }
        else {
            t.out.println("Hello from a Java CICS application");
        }
    }
}
```

4. Create your CICS bundle.

   Use the existing New CICS Bundle Project wizard to deploy your Java application. Use the New CICS OSGi Bundle wizard to decide which Eclipse plug-in projects to include and which JVM server will be used at run time.



Include OSGi Project in Bundle

Include OSGi Project in Bundle

Choose Project containing Java code to be included in the CICS bundle

Select the project to be included in the bundle

com.ibm.cics.server.examples.hello
com.ibm.cics.server.examples.jcics
com.ibm.cics.server.examples.web

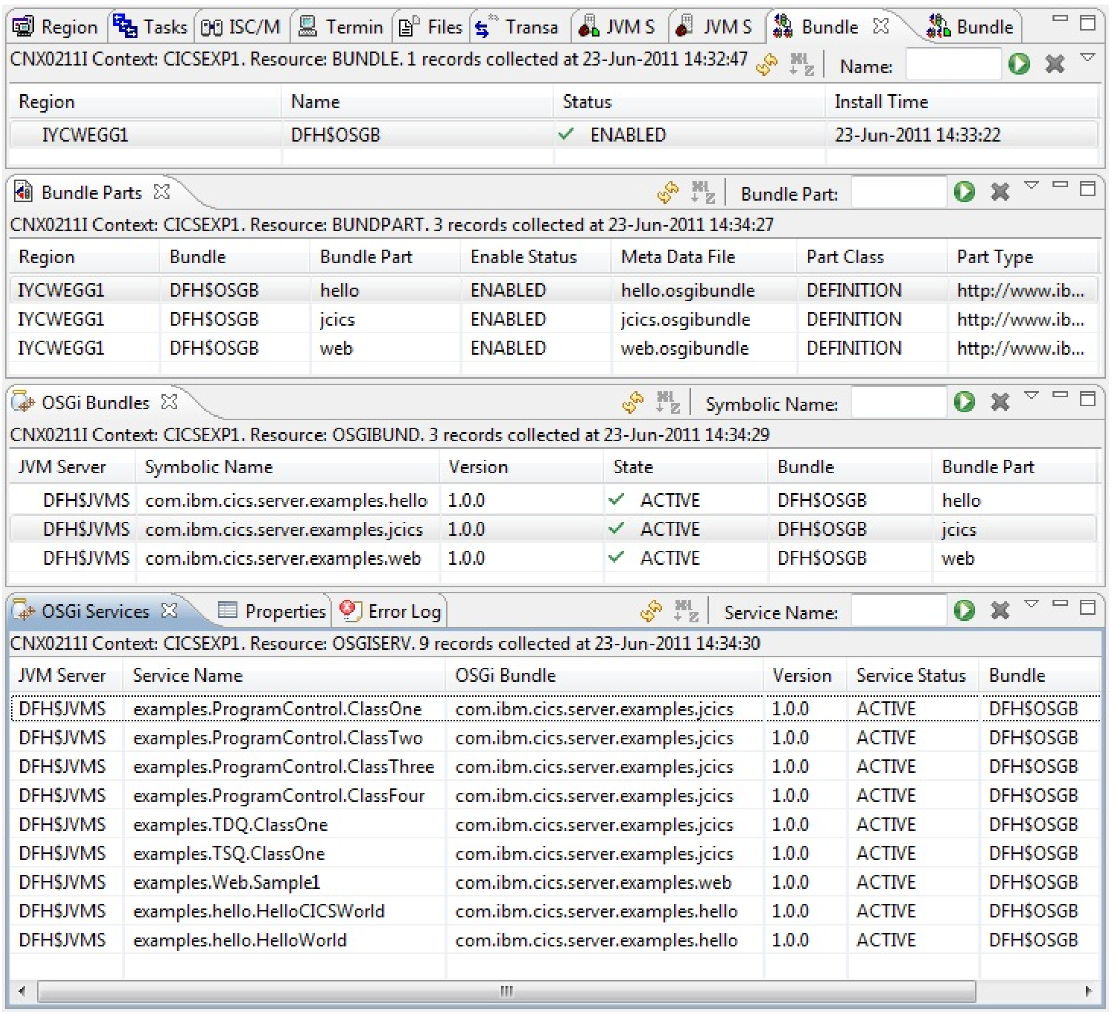
OSGi Bundle directive:

Symbolic Name: com.ibm.cics.server.examples.hello
Version: 1.0.0
JVM Server: DFH$OSGI (required)

File Name: hello.osgibundle
(Use the back button to change the name of the osgibundle file)

5. Deploy your CICS application.

   Select **Export to z/OS UNIX File System...** from the pop-up menu for your CICS bundle project to send it over to CICS and install it with a CICS bundle definition in your CSD or CICSPlex® System Manager BAS. Open the new OSGi Bundles and OSGi Services views to see if the application is ready to run. If there are any problems check the JVM trace file using the z/OS® UNIX Files view for any error messages.



CNX0211I Context: CICSEXP1. Resource: BUNDLE. 1 records collected at 23-Jun-2011 14:32:47

Region	Name	Status	Install Time
IYCWEGG1	DFH$OSGB	✓ ENABLED	23-Jun-2011 14:33:22

CNX0211I Context: CICSEXP1. Resource: BUNDPART. 3 records collected at 23-Jun-2011 14:34:27

Region	Bundle	Bundle Part	Enable Status	Meta Data File	Part Class	Part Type
IYCWEGG1	DFH$OSGB	hello	ENABLED	hello.osgibundle	DEFINITION	http://www.ib...
IYCWEGG1	DFH$OSGB	jcics	ENABLED	jcics.osgibundle	DEFINITION	http://www.ib...
IYCWEGG1	DFH$OSGB	web	ENABLED	web.osgibundle	DEFINITION	http://www.ib...

CNX0211I Context: CICSEXP1. Resource: OSGIBUND. 3 records collected at 23-Jun-2011 14:34:29

JVM Server	Symbolic Name	Version	State	Bundle	Bundle Part
DFH$JVMS	com.ibm.cics.server.examples.hello	1.0.0	✓ ACTIVE	DFH$OSGB	hello
DFH$JVMS	com.ibm.cics.server.examples.jcics	1.0.0	✓ ACTIVE	DFH$OSGB	jcics
DFH$JVMS	com.ibm.cics.server.examples.web	1.0.0	✓ ACTIVE	DFH$OSGB	web

CNX0211I Context: CICSEXP1. Resource: OSGISERV. 9 records collected at 23-Jun-2011 14:34:30

JVM Server	Service Name	OSGi Bundle	Version	Service Status	Bundle
DFH$JVMS	examples.ProgramControl.ClassOne	com.ibm.cics.server.examples.jcics	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.ProgramControl.ClassTwo	com.ibm.cics.server.examples.jcics	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.ProgramControl.ClassThree	com.ibm.cics.server.examples.jcics	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.ProgramControl.ClassFour	com.ibm.cics.server.examples.jcics	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.TDQ.ClassOne	com.ibm.cics.server.examples.jcics	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.TSQ.ClassOne	com.ibm.cics.server.examples.jcics	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.Web.Sample1	com.ibm.cics.server.examples.web	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.hello.HelloCICSWorld	com.ibm.cics.server.examples.hello	1.0.0	ACTIVE	DFH$OSGB
DFH$JVMS	examples.hello.HelloWorld	com.ibm.cics.server.examples.hello	1.0.0	ACTIVE	DFH$OSGB

If you make any changes just discard the CICS bundle and repeat step 5. If you want to share your application with colleagues use the Eclipse team support to check the relevant projects into Rational Team Concert, SVN, or any standard repository.

The Eclipse-based development environment offers a number of options for customers migrating Java applications from earlier versions of CICS, for those wishing to maintain co-existence with the pooled JVM run time or others simply taking advantage of the improved experience of using OSGi. Applications destined for JVM server must be deployed as OSGi bundles. Eclipse offers the following techniques to help:

- A few clicks can convert your existing Java project into a plug-in project. The Eclipse Plug-in Development Environment (PDE) checks for unsupported dependencies. You can still run the deployed code in a pooled JVM by adding the OSGi bundle JAR files to the class path in your JVM profile.
- Use the new project wizard to take an existing JAR file and automatically "inject" the required OSGi manifest file. The code does not need to be recompiled, reducing the need for extensive retesting. Use this technique if you want to redeploy to both pooled JVMs and JVM servers.
- Use the new project wizard to "wrapper" an existing JAR file and automatically add the required OSGi manifest file. Use this approach if licensing or other restrictions prevent artifact modification.

Included with the SDK is an Eclipse target platform template for every supported version of CICS TS, defining the correct level of JCICS and JRE. This ensures that application developers only use those Java APIs that are supported in the intended CICS server. Most importantly, whether your plans to move to JVM server are long-term or short-term, you can start taking advantage of OSGi today.

The Explorer SDK is aimed at both experienced Java application developers who are new to CICS and those with a CICS application development background. The plug-in extends the familiar Java developer UI with the documentation and tools necessary to target a CICS environment.

CICS system programmers who are evaluating Java support or working with application developers to get their applications running will find a full function CICS Explorer in the SDK. While the included samples are intended to show developers how to write CICS Java applications, no experience or understanding is required to create, deploy, and install them. Full access to the sample definitions and JVM server profiles is also available using the SDK.

Web services connectivity using Axis2

CICS TS 4.2 offers significant new options for hosting web services workloads. These improvements encompass three main areas:

- Reduction of costs through the use of z/OS Application Assist Processors (zAAP).
- Rapid web service development in Java through support for JAX-WS applications.
- Hosting of WSDL documents within CICS.

CICS TS 4.2 also extends the Web 2.0 Atom feeds capability from CICS TS 4.1 to offer a simplified approach for exposing CICS data sources to external users.

Reduction of costs through the use of zAAP

CICS TS 4.2 exploits an open technology called Axis2 in order to reduce the cost of hosting web services in CICS. Axis2 is a Java based project from the Apache Foundation, and is widely used for implementing SOAP support in Java based environments. CICS TS 4.2 implements a new deployment option for SOAP PIPELINE resources which allows much of the infrastructure processing to be performed in Java within a CICS JVM server. This new option uses Axis2, and is eligible to run on zAAP processors.

Switching an existing CICS pipeline to use the zAAP enabled Axis2 mode in CICS TS 4.2 involves a simple configuration change. For some workloads this can result in >10% of the workload being zAAP eligible. However, this potential cost saving comes at a cost of additional path length, so response times could suffer slightly. Not all applications will experience the same cost reduction results when switched to run under Axis2. The types of applications that will see the best results are those where the XML parsing and processing costs are especially high; for example, where there is a significant quantity of XML parsing, and where the XML contains a high degree of complexity. For simple applications the cost of switching into the Java environment can outweigh the benefit.

The use of Axis2 is optional. It is both simple to enable, and simple to disable. It is supported in both provider and requester modes, and requires no application changes. It uses the same web service bindings that are familiar from the existing web service support in CICS. If you have spare zAAP capacity and complex web services in CICS, then this could offer a significant cost reduction.

Rapid web service development in Java with JAX-WS

CICS TS 4.2 introduces a significant new application development option for developers who are comfortable working in Java. JAX-WS is a Java-based technology for writing web services in the Java programming language. It supports both bottom-up and top-down development styles, and is a regular part of Java 6.0. Provider mode JAX-WS applications can be hosted in the Axis2 environment within CICS. They run in the same JVM server as used by the Axis2 pipeline within CICS.

This close integration between the application and Axis2 environments ensures the most efficient use of the Axis2 environment, the most effective use of the zAAP processors, and the closest possible fidelity to application development styles used in other Java based hosting environments.

JAX-WS applications are a new programming model for CICS, and should not be confused with existing CICS web services concepts. JAX-WS applications can be hosted alongside web service binding based web services, but they are not the same thing. A JAX-WS application is always written in Java, it has a different deployment mechanism, and does not have a WEBSERVICE resource in CICS. This new concept provides a new option when considering web services for CICS; it does not interfere with any of the existing options.

JAX-WS can assist in scenarios where the CICS web services assistants are unable to support a complex WSDL document. Some WSDL documents use constructs that DFHWS2LS does not support; constructs such as recursion, or recurring model groups. In such scenarios the use of a JAX-WS based Java application could be considered as an alternative mechanism for implementing the web service. The JAX-WS generator is used to generate a Java template from the WSDL document. An application developer must then implement the appropriate business logic. The resultant Java program can interact with existing CICS assets through use of the JCICS API.

JAX-WS is an ideal technology for any organisation with Java programming skills and a requirement to host a web service in CICS.

Hosting of WSDL documents in CICS

A common concern for organisations deploying web services is the issue of WSDL proliferation and deployment. Each web service has an associated WSDL document to describe its programming interface. The problem of how to share that WSDL with client-side developers is something that each organisation addresses differently, as is the issue of how to ensure that the document is up-to-date with respect to any subsequent changes.

Use of a web services registry such as the WebSphere Service Registry and Repository (WSRR) can significantly assist with these scenarios, but there still remains a requirement to load the repository with the WSDL.

An informal protocol exists which allows a WSDL document to be hosted alongside the associated service. This allows a simple web browser to recover the WSDL using the URI at which the service itself is deployed. This protocol is implemented in, for example, WebSphere Application Server, and is something that customers have requested from CICS.

If the WSDL document is deployed to CICS along with the associated web service, then under CICS TS 4.2 an additional URIMAP resource is installed which will allow the associated WSDL document (or family of documents in a ZIP archive file) to be returned as a response to an HTTP GET request that targets the URI of the service, suffixed with “?wsdl”.

For example, if there is a service hosted in CICS at the following relative URI:

/services/exampleService/sales

Then CICS will return the associated WSDL document if it receives an HTTP GET request for the following relative URI:

/services/exampleService/sales?wsdl

This allows a simple method to query CICS for an up-to-date copy of the WSDL.

Summary – Conclusion

This paper has described how both significant evolutionary enhancements and fundamental architectural changes have dramatically raised the value proposition for developing CICS applications in Java. Not only does CICS provide the development environment and runtime support to reuse existing Java components to the benefit of existing and new CICS applications, but the reduction of costs through using the IBM System z Application Assist Processors (zAAP) and the IBM System z Integrated Information Processors (zIIP) make this an attractive workload to run in CICS.

The JVM server represents a move to a more industry standard Java experience in CICS to provide a server-side programming model where applications can share an 'engine' as well as sharing data between tasks. This greatly improves the potential for reuse of existing Java components.

Support for the OSGi standard means that applications written for the JVM server or migrated from pooled JVMs can be packaged as OSGi bundles to enable dynamic deployment, replacement, and versioning of applications within the JVM server without needing to restart the JVM.

The skills required to develop, debug, and monitor Java applications are based around familiar tools, such as Eclipse, CICS Explorer, and Rational Application Developer for System z. The availability of standard tools for debugging, problem determination, and performance monitoring enhances skills reuse and knowledge transfer.

The advantages of the JVM server also extend to web services, through the support for JAX-WS applications.

The dramatic enhancements to the Java support in CICS TS 4.2 represent a game-changing shift in the technology used to run Java applications in CICS. There has never been a better time to consider using Java applications in CICS TS.

Further reading

1. Java support in CICS:
   http://publib.boulder.ibm.com/infocenter/cicsts/v4r2/topic/com.ibm.cics.ts.java.doc/JVMserver/JVMsupport.html

2. CICS Explorer downloads, including the SDK:
   http://www.ibm.com/software/htp/cics/explorer/download/