IBM – CICS SupportPac CS1S

Using User-Defined BUNDLEs to Control CICS Dynamic Scripting Applications

Version V1.0.0.0

September 25th, 2011

First Edition –September 25th, 2011 - SupportPac CS1S release V1.0.0.0

Please send any questions or comments to:
Dennis Weiand, IBM Corporation
1503 LBJ Freeway
Dallas, TX 65234

Note: Before using this information and the product it supports, read the information in “Notices” on page 4.

Contents

The typical CICS Dynamic Scripting application is controlled (start/stop) using line commands. For a production environment, you may wish to have your CICS Dynamic Scripting application available at CICS region startup instead of requiring operators to use line commands (from TSO or a TTY terminal). The goal of this SupportPac is to offer a technique that can be used to make CICS Dynamic Scripting applications available at CICS startup.

Notices

Trademarks

Section 1: Overview

One of the design points of CICS Dynamic Scripting was to make the CICS implementation of the Project Zero technology in the CICS Transaction Server Feature Pack for Dynamic Scripting look as much as possible like any other Project Zero implementation. This means the primary way to administer a CICS Dynamic Scripting application (start and stop) is through a command line interface.

While this is a benefit for programmers and administrators that are familiar with Project Zero technology, this does deviate from the methods normally used to control CICS-based production applications. Typical CICS applications can be made available at CICS startup without operator intervention. Most CICS customers include a CICS region startup as part of their z/OS IPL procedures and no further actions are necessary to make a production application available.

This SupportPac offers a technique that can be used to make CICS Dynamic Scripting applications available at CICS region startup.

In this SupportPac we describe and implement a way of using a CICS User-Defined BUNDLE Resource to control the start and stop operations for a CICS Dynamic Scripting application. You can use the CICS Explorer, CEDA, or any other CICS-provided way of defining and installing CICS BUNDLE resources.

The ‘install’ or ‘enable’ of this User-Defined Bundle resource starts the specified CICS Dynamic Scripting application, and a ‘disable’ of the resource will stop the CICS Dynamic Scripting application.

The BUNDLE resource that describes your CICS Dynamic Scripting application can be placed in a group and added to your startup list, which then causes the CICS Dynamic Scripting application to be started at CICS region startup.

To implement this technique for starting/stopping a CICS Dynamic Scripting application you will need to:

- Configure your CICS Dynamic Scripting Application owning CICS region for Java
- Add a .jar file in your JVMProfile file’s CLASSPATH_SUFFIX
- Define 3 programs to CICS (one as a PLTPI program)
- Expand a PDS containing two COBOL programs, and add the PDS to your DFHRPL
- Create BUNDLE resource(s) for your CICS Dynamic Scripting applications

Note that the technique used in this SupportPac has only been tested using with z/OS and CICS running in code page IBM-037.

Also note that as of the date of this paper CICS Dynamic Scripting is only available with CICS TS V4.1, although there is a statement of direction for the CICS Transaction Server Feature Pack for Dynamic Scripting to be added to CICS TS V4.2.

1.1 CICS Dynamic Scripting Application Administration

Administration of CICS Dynamic Scripting applications is normally performed using line commands known as the command line interface (CLI). These commands are also known as ‘zero’ commands since CICS Dynamic Scripting is an implementation of the Project Zero technology, and ‘zero’ is the name of the actual program that is being invoked. The command line interface provides for various administrative functions that are needed to control a Dynamic Scripting application. Although there are several commands, the primary commands used to control a Dynamic Scripting application are ‘zero start’, ‘zero stop’, ‘zero resolve’, ‘zero update’, and ‘zero create’. When using the ‘zero start’ and ‘zero stop’ commands, you would normally use a TTY session with z/OS’s UNIX System Services (USS) or a TSO OMVS session with USS, change to your application’s home directory, and issue the appropriate ‘zero’ command.

1.2 CICS BUNDLE Resources

In CICS TS V4.1 (available June 2009), a BUNDLE resource was added to CICS. In addition to being used for SCA, Event Processing, and TRANSFORMs, the BUNDLE resource is extensible. This means that you can extend BUNDLEs to control your own resource types. You can have CICS invoke code that you write when your resource is installed, enabled, disabled, inquired, or discarded.

To tell CICS about the existence of your resource, you write a ‘register program’ that tells CICS that your resource exists and that when actions are taken against your resource (install, enable, inquire, disable, discard), a specified ‘callback’ program should be invoked.

To ‘define’ occurrences of your new resource to CICS, you create a BUNDLE resource. A BUNDLE resource points to a directory on z/OS’s USS. Your BUNDLE directory contains a META-INF/cics.xml file called the ‘manifest’. The manifest is an XML file that describes the resources in the bundle and their dependencies. For this SupportPac, the XML tag of most importance in the manifest file is the ‘define’ tag. The define tag allows you to define the ‘name’, ‘type’, and ‘path’ for your resource. The ‘name’ specifies the name of your resource, the ‘type’ indicates that it is the new resource type you just registered with CICS, and the ‘path’ indicates the name of a file in the BUNDLE’s root directory that contains further information about your resource. Multiple ‘define’ tags are allowed in a manifest, so you can describe multiple resources in a BUNDLE. When the BUNDLE is installed, all the resources defined (with corresponding define tags) in the manifest are installed. For your user-defined resource, installation of the BUNDLE tells CICS to invoke the callback program specified by the resource registration program. CICS invokes the registered callback program passing it a parameter list that indicates the requested action (i.e. create, enable, disable, inquire, and discard). The registered program is responsible for taking the appropriate action, and telling CICS the results.

1.3 Combining BUNDLEs and CICS Dynamic Scripting Applications

There are several resource types already defined to CICS that can be specified in the type= attribute of the define tag. If you are going to create your own resources you need to tell CICS about your new resource in a registration program. The registration program tells CICS the name of the new resource type and specifies the name of a callback program that should be invoked when actions are requested against the resource (like install, enable, etc). As indicated above, when a BUNDLE is installed that defines an instance of your new resource, CICS invokes the registered callback program (that you specified when you registered your new resource type). The callback program is expected to perform the requested action on the resource (create, enable, disable, inquire, and discard), report the success or failure of the action, and report the current status of the resource.

For purposes of this SupportPac, we have written a program (named DDWREGIS) that registers a 'ZEROAPP' resource with CICS and indicates that if there is a request to create, enable, disable, inquire, or discard the 'ZEROAPP' resource, that a program named DDWZAPPU should be invoked (the DDWZAPPU name was chosen for Zero APPlication status Update). The register program is normally specified in the PLTPI so that your new resource type is available for actions during CICS startup.

For this SupportPac, the name of the user-defined resource type is in the same style of other resources CICS provides, with the lowest level part (trailing characters) being 'ZEROAPP'.

When the 'callback' program (DDWZAPPU in this case) is invoked for a 'create', CICS passes a channel with various containers to the DDWZAPPU program. A control container indicates the function (e.g. create), and the desired state (e.g. enabled). There is also a container that holds the 'name' of the resource (the resource name is specified in the define tag of the BUNDLE manifest), a container that holds the resource type (ZEROAPP in this case), a container that holds the name of the BUNDLE directory, and a container that holds the contents of the file (in the BUNDLE directory) that was specified in the path= attribute of the define tag in the manifest file. The callback program (DDWZAPPU in this case), for a create, is responsible for placing a token in the control container along with creating the resource and changing the resource to the requested state. The token is used for subsequent requests from CICS relative to this resource.

When CICS invokes the callback program for enable, disable, inquire, and discard (i.e. requested actions other than create), CICS only provides a control container. The control container holds the token you placed into the control container during the create invocation, plus an indication of the desired state.

For this SupportPac, the registration program (DDWREGIS) registers a resource named "`http://www.ibm.com/xmlns/prod/cics/bundle/ZEROAPP`" that is associated with the callback program DDWZAPPU.

To define a CICS Dynamic Scripting application, you would place a 'define' tag in the BUNDLE manifest as follows:

```
<define name="ZeroApp1" type=http://www.ibm.com/xmlns/prod/cics/bundle/ZEROAPP
path="ZeroApp.ascii"/>
```

(the above is all on one line)

When the BUNDLE resource referencing the directory containing this manifest is installed, CICS invokes the DDWZAPPU program providing the containers listed above, and specifies a function of ‘create’ and a status of ‘enabled’.

For functions other than create, only a control container (token, function, and state) is provided, so DDWZAPPU stores the information provided during the create in a TS Queue. I use a 16 character TSQueue name. The first 8 characters are something unique for DDWZAPPU, and the second 8 are the resource name. Additionally, during a create, DDWZAPPU places the resource name in the ‘client token’ (the 8 characters) that is provided during other function requests. **Note** that this means that I have imposed an artificial limitation on the size of a ZEROAPP resource name of 8 characters. I could have used other techniques, but using the resource name for both the client token and second part of the TSQ provided an easy programming approach. Also during a create, I create a Korn shell script in the BUNDLE’s root directory (more on this script a bit later).

To control your CICS Dynamic Scripting application, DDWZAPPU invokes a Korn shell script to do a ‘zero start’ or ‘zero stop’. CICS needs to know the application’s home directory (the Korn shell script must be invoked from this location), and the name of a file that contains the environment variable values that are specific to your application’s environment. For this implementation, I have you create a file in the BUNDLE’s root directory that has an APP_HOME= parameter to reference the applications home directory and a SETENV= parameter to reference a script that sets variables for the application’s environment. The file containing these to values is placed in the BUNDLE’s root directory and is named in the path= of the define tag in the BUNDLE’s manifest. This file must have a UTF-8 encoding (same as the manifest file).

When DDWZAPPU receives an ‘enable’ request, it invokes a Java program named “DDWZEROJ” passing the application’s home directory, the name of the environment setup file, and the characters ‘start’ to the Java program.

During resource ‘create’, I create a Korn shell script named DDWZeroCommand.sh in the BUNDLE’s root directory (in the local code page) which looks as follows:

```
#!/bin/ksh
cd $1
. $2
zero $3
```

The DDWZEROJ Java program invokes the DDWZeroCommand.sh script passing it the application’s home directory (which is picked up as $1), the name of the file containing environment variables that establish your application’s environment ($2), and ‘start’ ($3).

The outcome of invoking Korn shell script (i.e. the ‘zero start’) is reported back to DDWZAPPU. If successful, DDWZAPPU indicates a status of ‘enabled’. If it fails, DDWZAPPU indicates a status of ‘disabled’. Note that although it might be logical to indicate a status of ‘failed’, I chose not to report a ‘failed’ condition since a ‘failed’ indication will cause CICS to believe that the resource is ‘unusable’, in which case the callback program is never invoked for that resource again (unless you discard and create the resource again).

For a ‘disable’, DDWZAPPU again LINKs to the DDWZEROJ Java program, which again invokes the DDWZeroCommand.sh Korn shell script. The only difference is that the third parameter for the DDWZeroCommand.sh script is ‘stop’.

For an ‘inquire’, DDWZAPPU and DDWZEROJ do a quick check of some files in your CICS Dynamic Scripting application’s home directory and CICS resources to determine whether the application is running. If the application is running, DDWZAPPU reports ‘enabled’, and if your application is not running DDWZAAPU reports ‘disabled’. This uses an undocumented/unsupported technique to determine if the application is running, so unfortunately, if the files used to do this quick check are no longer used/provided by CICS Dynamic Scripting in a future release, the ‘inquire’ function in DDWZAPPU will have to be changed (i.e. re-coded).

I already knew how to invoke Korn shell scripts and looking at various directories and files in z/OS UNIX System Services using Java, so that is why I wrote DDWZEROJ in Java.

Note that the requirements for the contents of the file containing environment variable settings for this SuportPack are the same environment variable requirements as documented in the CICS InfoCenter. See step 4 at
http://publib.boulder.ibm.com/infocenter/cicsts/v4r1/index.jsp?topic=/com.ibm.cics.ts.smash.doc/smash_installation.html

1.4 Recommendations

1. I recommend **one** CICS Dynamic Scripting application per BUNDLE so that you can start and stop your CICS Dynamic Scripting applications individually. All CICS Dynamic Scripting applications in the same BUNDLE resources will be started and stopped at the same time. You are likely to run into situations where the CICS Dynamic Scripting applications will need to be started and stopped individually. Since you can have as many BUNDLE resources as you would like, you could have a resource group containing several BUNDLE resources, and when starting the group, the CICS Dynamic Scripting applications defined in each BUNDLE in the group will be started. If you need to later stop one of your CICS Dynamic Scripting applications, you can disable its associated BUNDLE resource individually. The programs in the SupportPac will allow you to define multiple CICS Dynamic Scripting applications in a single BUNDLE resource if that is your preference.

2. It is **strongly** recommended that you create your script containing environment variables and their settings, and **test** starting and stopping of your CICS Dynamic Scripting application manually before starting and stopping it via the technique described in this SupportPac. If the 'zero start' or 'zero stop' command fails for any reason, the callback program will report the status as disabled. If your CICS Dynamic Scripting application cannot start, it is likely that problem determination will be easier while manually invoking the script to set the environment variables and manually performing the 'zero start' and 'zero stop'.

   To test your script that sets environment variables, you can:

   - Create the script

   - From an OMVS command line invoke the script using the following
   `. <your_script>` (that's dot space and then the name of your script)

   The dot space says to run the script in the current process. Without the dot space, running your script will have no affect.

   - Then, from OMVS, from your application's home directory, use the 'zero start' command.

   - If you can start your application this way, you should be able to use my BUNDLE façade to start and stop your application

3. The supplied programs try to ensure that the CICS Dynamic Scripting application whose home directory you specify is intended to run in the same region as where you are installing the BUNDLE resource. The program does this by looking at the NETNAME= parameter in your application's config/zerocics.config file (or your CICS Dynamic Scripting installation's config/zerocics.config file if your application doesn't contain this file), and checking to see if there is a connection definition with the same NETNAME in the current CICS region. Error messages are issued if appropriate.

Section 2: Relationship of Parts

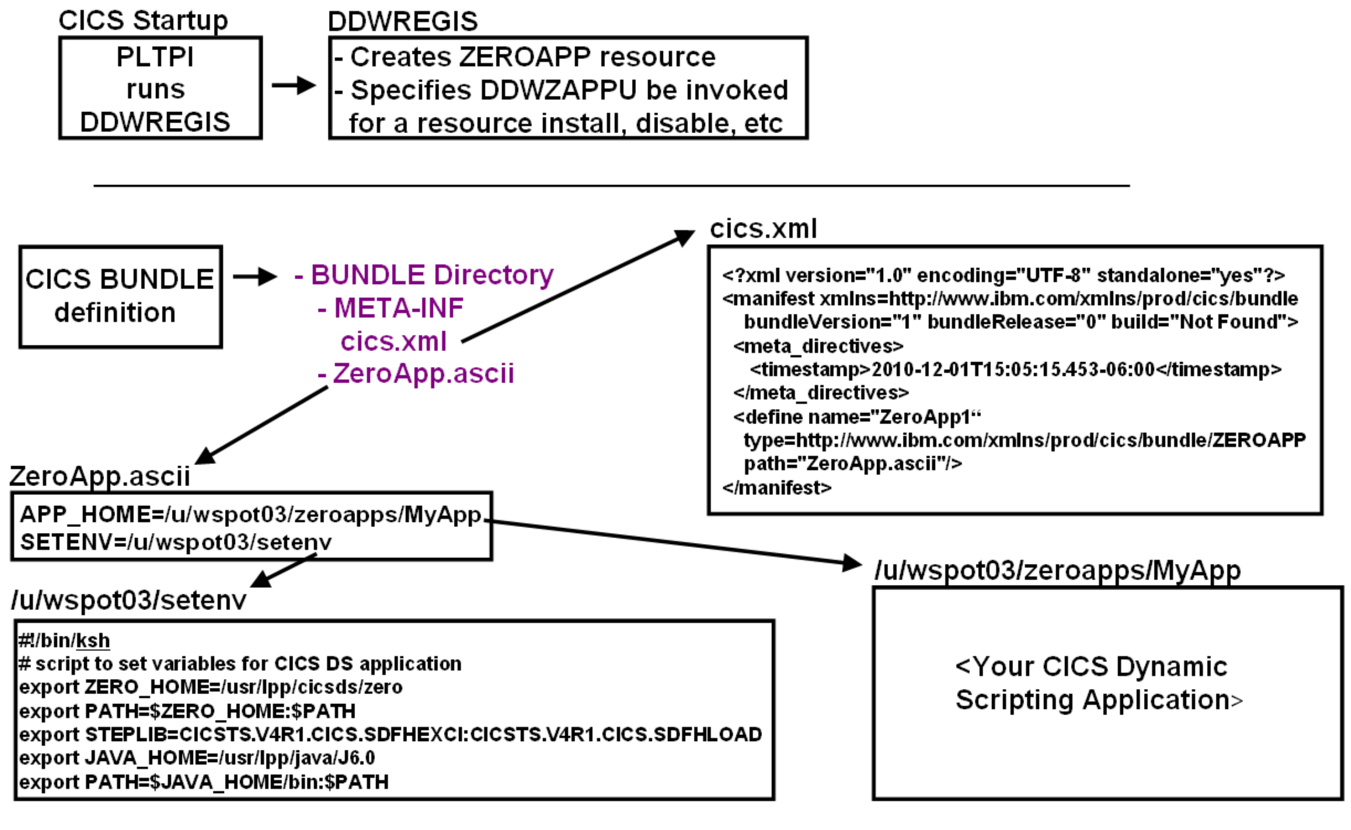
The diagram below shows the relationship between the resource definitions and the associated directories and files.

At CICS startup, as part of the PLTPI, CICS invokes the DDWREGIS program. The DDWREGIS program creates a resource named ZEROAPP and tells CICS that when a BUNDLE that specifies this resource is installed, enabled, etc, that the DDWZAPPU program should be invoked.

You create a BUNDLE resource for each of your CICS Dynamic Scripting applications. The BUNDLE resource points to a BUNDLE directory containing a META-INF/cics.xml and a ZeroApp.ascii file. The META-INF/cics.xml file specifies that a resource type of ZEROAPP is to be installed, the resources name, and the name of a resource file (ZeroApp.ascii).

The BUNDLE installation causes CICS to invoke the DDWZAPPU program which looks at the ZeroApp.ascii file to find the location of your CICS Dynamic Scripting application, and the name of a Korn shell script that can be invoked to establish environment variables so that the CICS Dynamic Scripting Application can be accessed.

The DDWZAPPU program invokes the Korn shell script to establish the environment variables, and then invokes a 'zero start' command to start the specified CICS Dynamic Scripting application.


CICS Startup
PLTPI runs DDWREGIS
DDWREGIS
- Creates ZEROAPP resource
- Specifies DDWZAPPU be invoked for a resource install, disable, etc
CICS BUNDLE definition
- BUNDLE Directory
- META-INF
cics.xml
- ZeroApp.ascii
cics.xml
ZeroApp.ascii
/u/wspot03/setenv
/u/wspot03/zeroapps/MyApp
<Your CICS Dynamic Scripting Application>

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<manifest xmlns=http://www.ibm.com/xmlns/prod/cics/bundle
   bundleVersion="1" bundleRelease="0" build="Not Found">
  <meta_directives>
    <timestamp>2010-12-01T15:05:15.453-06:00</timestamp>
  </meta_directives>
  <define name="ZeroApp1“
   type=http://www.ibm.com/xmlns/prod/cics/bundle/ZEROAPP
   path="ZeroApp.ascii"/>
</manifest>
```

```
APP_HOME=/u/wspot03/zeroapps/MyApp
SETENV=/u/wspot03/setenv
```

```
#!/bin/ksh
# script to set variables for CICS DS application
export ZERO_HOME=/usr/lpp/cicsds/zero
export PATH=$ZERO_HOME:$PATH
export STEPLIB=CICSTS.V4R1.CICS.SDFHEXCI:CICSTS.V4R1.CICS.SDFHLOAD
export JAVA_HOME=/usr/lpp/java/J6.0
export PATH=$JAVA_HOME/bin:$PATH
```

Figure 1.0

Section 3: Installation

The technique described in this SupportPac uses a registration program (named DDWREGIS) to tell CICS about the ZEROAPP resource and to indicate that any function requests (create, enable, disable, inquire, discard) for this resource should be sent to the DDWZAPPU program. The DDWZAPPU program (written in COBOL) LINKs to a Java program named DDWZEROJ when DDWZAPPU needs its services.

3.1 Prerequisites

This SupportPac has prerequisites of CICS TS V4.1 and the CICS Transaction Server Feature Pack for Dynamic Scripting. As of the date of this paper, the CICS Transaction Server Feature Pack for Dynamic Scripting is only available in CICS TS V4.1. There is a statement of direction for the CICS Transaction Server Feature Pack for Dynamic Scripting to be available with CICS TS V4.2 at some point.

See the CICS InfoCenter CICS Dynamic Scripting installation instructions.

3.2 SupportPac Installation Steps

The steps below have you define three programs to CICS. You may want to place these three programs in a group by themselves and place this group at the top of your region startup group list.

This part only needs to be done once per BUNDLE-owning CICS region.

___1. **Unzip** the **cs1s-v1.0.0.1.zip** file.

___2. Use the File Transfer Program (**FTP**) to transfer the files to z/OS

a. Transfer the **com.ibm.ddw.java.util.cicsds.jar** file to the z/OS UNIX file system in binary mode.

b. Transfer the **cs1s.unload** file as a z/OS sequential file as a z/OS sequential using the following FTP commands (where userid is replaced with your own z/OS userid):

```
cd //userid
binary
quote site blksize=3120 lrecl=80 recfm=FB
put cs1s.unload
```

___**3.** Use the TSO RECEIVE command to reload the cs1s.unload file

```
RECEIVE INDATASET(CS1S.UNLOAD)
```

Note that at the prompt you can specify
DSNAME('THE.RESTORED.PDS.NAME.YOU.WANT')

Note that entering the above command reconstructs the CS1S.LOAD library.

___**4.** **Add** the **CS1S.LOAD** dataset to your CICS region's **DFHRPL**.

___**5.** **Configure Java support** in your CICS region (for the DDWZEROJ program). See the CICS InfoCenter if you are not sure how to configure Java in your CICS region.

___**6.** **Add the com.ibm.ddw.java.util.cicsds.jar file** included with this SupportPac to your JVMProfile file, in the CLASSPATH_SUFFIX= parameter.

___**7.** **Define** the Java **program** to CICS

Parameter	Value
Name	DDWZEROJ
Group	DDWZROGP
EXECKey	Cics
Concurrency	Threadsafe
JVM	Yes
JVMClass	com.ibm.ddw.java.util.cicsds.DDWCicsDSAppStartStopStatus
JVMProfile	DFHJVMPR (you can use any name you want for your JVMProfile file, but you will need to add the JAR file specified above to your CLASSPATH_SUFFIX=, see note above)

If you define the Java program to CICS specifying the JVMProfile file DFHJVMPR (the default JVMProfile file supplied with CICS), you will need to change the CLASSPATH_SUFFIX= in that file to include the directory where you placed the com.ibm.ddw.java.util.cicsds.jar file (see step #2 above).

___**8.** **Define** the DDWREGIS **program** to CICS

Parameter	Value
Name	DDWREGIS
Group	DDWZROGP
Language	Cobol
EXECKey	Cics

___**9.** **Define** the **DDWREGIS** as a **PLTPI** program after DFHDELIM.

```
DFHPLT TYPE=ENTRY,PROGRAM=DDWREGIS
```

___**10. Define** the **DDWZAPPU** program to CICS (this program uses the CICS SPI)

Parameter	**Value**
Name	DDWZAPPU
Group	DDWZROGP
Language	Cobol
EXECKey	Cics
Api	Openapi

___**11. Add** the **DDWZROGP group to** the top of your **startup list**.

Section 4: Define a BUNDLE to control your CICS Dynamic Scripting Application

For <u>each</u> CICS Dynamic Scripting application you want to control using CEDA/CEMT, the CICS Explorer, or CPSM, you will need to define a BUNDLE resource and the corresponding BUNDLE directory and manifest.

___**1.** From the **CICS Explorer or RDz** (Rational Developer for System z), switch to or open a **Resource perspective**.

___**2.** From the **Resource perspective**, the **Project Explorer view**, create a **CICS Bundle Project** named **CICSBundleZeroApp1**.

Note that you could use any name you want for the name of the BUNDLE project, however these directions will assume you used the name CICSBundleZeroApp1.

___**3. Expand** the **CICSBundleZeroApp1** project.

Note that you will see a directory in the project named META-INF. If you expand the META-INF directory, you will see a file named cics.xml. This is the bundle manifest file.

___4. In the **CICSBundleZeroApp1** project define a file named **ZeroApp.ascii**.

Note that you can name this file anything you want, but whatever name you use will need to be specified in the path= parameter in the manifest. (see later step)

Note that we chose the .ascii extension because RDz has a default transfer type for this extension of binary. You can use any extension you want, but this file must be in same code page as Windows runs in, or the file can be in UTF-8. The manifest file must also be transferred as binary. RDz has a default transfer type of binary for .xml files, but you may want to check on the transfer type on your RDz system (at Menu Bar->Window->Preferences, Remote Systems->Files on the left, and transfer types will be on the right).

Note that if you transfer your bundle project to z/OS using the CICS Explorer (the Export to z/OS HFS option), everything is transferred as binary (which is the correct thing to do for this BUNDLE project).

___5. **Edit** the **ZeroApp.ascii** file to include the following:

```
APP_HOME=<your_CICS_DS_application_home_directory>
SETENV=<name_of_your_Korn_shell_script_to_set_environment_variables>
```

Below is the contents we used during our testing (MyApp is the name of our CICS Dynamic Scripting application)….

```
APP_HOME=/u/wspot03/zeroapps/MyApp
SETENV=/u/wspot03/setenv
```

___6. Ensure your **environment variable file** referenced by the SETENV= is appropriate.

The CICS InfoCenter describes the environment variables you need to set to run your CICS Dynamic Scripting application (the exact InfoCenter article was listed previously in this PDF).

You will want to create and manually test this file before using it with the technique/programs described in this SupportPac.

Below is an example of the environment file we used for our testing. The environment variables below are described in the CICS InfoCenter, although we should probably note that the directory set in the ZERO_HOME environment variable is the CICS Dynamic Scripting installation directory.

```
#!/bin/ksh
# script to set variables for the CICS Dynamic Scripting Feature Pack
export ZERO_HOME=/usr/lpp/cicsds/zero
export PATH=$ZERO_HOME:$PATH
export STEPLIB=CICSTS.V4R1.CICS.SDFHEXCI:CICSTS.V4R1.CICS.SDFHLOAD
export JAVA_HOME=/usr/lpp/java/J6.0
export PATH=$JAVA_HOME/bin:$PATH
```

___**7.** **Edit** the **META-INF/cics.xml** file (the bundle manifest) to define your CICS Dynamic Scripting application.

Change the contents of the manifest file as follows:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<manifest xmlns="http://www.ibm.com/xmlns/prod/cics/bundle" bundleVersion="1"
bundleRelease="0" build="Not Found">
<meta_directives>
  <timestamp>2010-12-01T15:05:15.453-06:00</timestamp>
</meta_directives>
<define name="ZeroApp1" type="http://www.ibm.com/xmlns/prod/cics/bundle/ZEROAPP"
path="ZeroApp.ascii"/>
</manifest>
```

Note that the initial manifest tag is on one line. **Note** that the define tag should be on one line

On the define tag, the value for name= is the name of the resource that represents your CICS Dynamic Scripting application. I have written the programs that come with this SupportPac to limit the maximum size of the name to 8 characters.

The value in the path= attribute is the name of the config file containing your application's home directory and the name of the file you are using to set your environment variables. See steps 5 and 6 in this section.

Note that you can have multiple define tags per manifest although I recommend you only define one CICS Dynamic Scripting application per BUNDLE resource.

___**8.** **Transfer** your **bundle project to z/OS** in the z/OS UNIX file system.

___**9.** **Define** a **BUNDLE** resource in CICS as follows:

Parameter	**Value**
Name	ZEROAPP1
Group	MYZROAPS
BUndledir	/u/wspot03/cicsbundles/CICSBundleZeroApp1
BAsescope	http://sample.org/zeroappservice

Note that for the above, the value for the BUndledir is the location of your bundle directory. In these directions, we had you create a bundle project named CICSBundleZeroApp1. So for the above, we transferred our CICSBundleZeroApp1 project to the /u/wspot03/cicsbundles directory.

Note that you will want to install/enable/disable/discard your BUNDLE resource to ensure it works properly before you add the group containing the BUNDLE resource to your startup list. My program does many checks to ensure it has all artifacts it needs and that they are in the proper status, so be sure to look at the MSGUSER DD statement in your CICS JCL if it doesn't work as expected. Actually I write messages to the CSMT TD destination, which most customers direct to the MSGUSER DD statement.

Section 5: Controlling your Application using the Bundle

Note that the DDWREGIS program must run before you attempt to install the bundle containing your ZEROAPP resource. In the above installation directions we asked you to add this program to the PLIPI, although for testing the SupportPac, you may find it easier to invoke the DDWREGIS program using CECI LINK.

If the BUNDLE resource that you defined in part 2 is placed in a group in your CICS region startup group list and the DDWREGIS program is specified in the PLTPI table as documented earlier, your CICS Dynamic Scripting application will be started during CICS startup.

If you don’t include the resource in your Region’s startup group list, you can still install the BUNDLE resource later, but you must be sure to run the DDWREGIS program first.

When the BUNDLE is installed, your CICS Dynamic Scripting application will be started and the status of your BUNDLE will be set to ‘enabled’.

If you disable your BUNDLE, your CICS Dynamic Scripting application will be stopped. If you enable your BUNDLE, your CICS Dynamic Scripting application will be started. As usual, you will have to disable your bundle before you can discard the resource.

If your CICS Dynamic Scripting application is manually started before the BUNDLE is installed (i.e. a ‘zero start’ from the command line) and you attempt to install the BUNDLE, the DDWZAPPU program will recognize that your application is already running and return a status of ‘enabled’. A subsequent BUNDLE disable will stop the application.

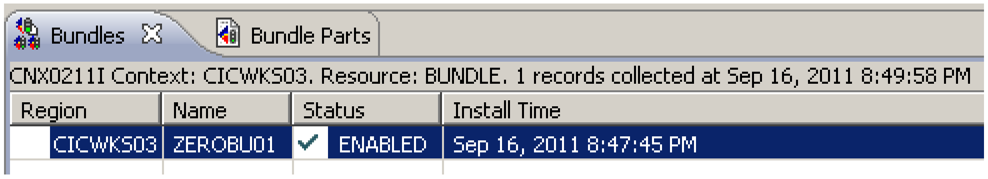
If your CICS Dynamic Scripting application is manually stopped after the BUNDLE is installed (i.e. a ‘zero stop’ from the command line), a status display of the BUNDLE will show disabled. If you then change the status to enabled (e.g. from CEMT or the CICS Explorer), your CICS Dynamic Scripting application will be started.

If you have multiple CICS Dynamic Scripting applications defined in the same BUNDLE, when the BUNDLE is installed, enabled, or disabled, all CICS Dynamic Scripting applications defined in the BUNDLE will be stopped/started. For ease of use, have one CICS Dynamic Scripting application per BUNDLE and multiple BUNDLEs in a group. You can install the group if you want all CICS Dynamic Scripting applications started.

Note that it takes a while for a CICS Dynamic Scripting application to be started and/or stopped. Your CEDA/CEMT or CICS Explorer display will wait for the operation to complete before the new status is displayed. You can look at your CICS MSGUSER JCL DD statement output for messages.

The user-define BUNDLE façade is intended for use in production and only the equivalent of a ‘zero start’ and ‘zero stop’ are provided.

After the BUNDLE is installed, you can display the BUNDLE status. For example:

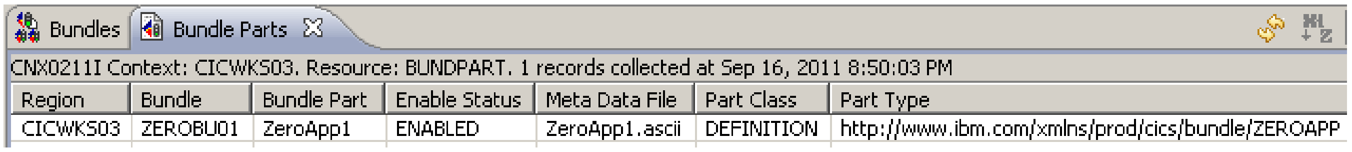


Bundles | Bundle Parts

CNX0211I Context: CICWKS03. Resource: BUNDLE. 1 records collected at Sep 16, 2011 8:49:58 PM

Region	Name	Status	Install Time
CICWKS03	ZEROBU01	✓ ENABLED	Sep 16, 2011 8:47:45 PM

You can also display the Bundle Parts which displays the values from the parameters on the define tab in the META-INF/cics.xml manifest file: the part type (ZEROAPP), the part name (ZeroApp1), and the path= file (ZeroApp.ascii).



Bundles | Bundle Parts

CNX0211I Context: CICWKS03. Resource: BUNDPART. 1 records collected at Sep 16, 2011 8:50:03 PM

Region	Bundle	Bundle Part	Enable Status	Meta Data File	Part Class	Part Type
CICWKS03	ZEROBU01	ZeroApp1	ENABLED	ZeroApp1.ascii	DEFINITION	http://www.ibm.com/xmlns/prod/cics/bundle/ZEROAPP

Section 5: Problem Determination

It is strongly recommended that you test the Korn shell script containing your environment variables manually before you try to implement my BUNDLE facade.

The DDWREGIS program must run before you attempt to install the bundle containing your ZEROAPP resource. In the above installation directions we asked you to add this program to the PLIPI, although for testing the SupportPac, you may find it easier to invoke the DDWREGIS program using a CECI LINK.

You may run into security issues related to your USS files (the bundle directory and manifest file), however the DDWZEROJ program does access checking so any security related issues **should** be detected and listed in the MSGUSER DD of your CICS region JCL (the programs write to the CSMT TD queue).

The DDWZAPPU program uses the SPI to do an INQUIRE on JVMServers and an INQUIRE on NETNAME. The DDWZAPPU program will be run under the CEDA, CEMT, or CSSY transaction (depending on where and when the BUNDLE install is requested), so this shouldn't provide a security problem.

If you define a TSQ named DDWZMSGS (put anything in it), the DDWZAPPU and DDWZEROJ programs will write many more messages to the CSMT destination which may be useful for debugging problems. Again, manually check your scripts before using this user-defined BUNDLE façade and you shouldn't need to turn on extra messages.

If the DDWZeroCommand.sh already exists in the bundle directory, DDWZEROJ will **not** overwrite it. This means that you can make your own DDWZeroCommand.sh and add additional debugging statements. Be aware that the DDWZeroCommand.sh must be in codepage 037 or 1047 since it is executed as a Korn shell script. Everything else associated with the bundle needs to be in UTF-8. It is recommended that you let me create the DDWZeroCommand.sh script, although if you are having problems and want more messages produced by the script, you could add a "set –x" at the beginning of the script.

Section 6: References

For more information on User-Defined Bundles, see the CICS TS V4.1 InfoCenter. Search for 'user resource bundle'.

For more information on controlling your CICS Dynamic Scripting application, see the CICS TS V4.1 InfoCenter and the Project Zero documentation (see projectzero.org).