IBM® WebSphere® Developer Technical Journal

to go!

Issue 14.1
February 2, 2011

Inside:

Plus:

IBM

IBM WebSphere Developer Technical Journal

Issue 14.1 : February 2, 2011

From the editor

This issue of the **IBM® WebSphere® Developer Technical Journal** has information to help you modernize your legacy 3270 applications, vary your reporting output to satisfy regulatory requirements, and build a smartphone application that integrates IBM WebSphere CloudBurst™ capabilities. Also, **The Support Authority** explains why testing in production is a bad idea, **Innovations within reach** describes a way to improve performance wth IBM WebSphere DataPower® Appliances, and **The WebSphere Contrarian** updates the discussion on virtualization.

Your required reading begins below…

Feature articles

- **Re-engineering applications using optimized local adapters on WebSphere Application Server for z/OS**
  by James T Mulvey, Timothy Kaczynski and Frank Golazeski
  IBM® WebSphere® Application Server for z/OS® V7 introduced optimized local adapters support to provide an optimized, high performance bi-directional connector for WebSphere Application Server and applications running in external z/OS address spaces. Optimized local adapters provide support for two-way synchronous local calling between applications in WebSphere Application Server and z/OS batch, CICS®, IMS™, ALCS, and USS processes. Optimized local adapters achieve optimal performance and very low latency by extending the WebSphere Application Server for z/OS local communications feature, which is based on z/OS cross memory services. It provides security context propagation in each direction with identity assertion, and it provides two-phase commit transaction support for calls between WebSphere Application Server and CICS. This article provides insight into how optimized local adapters work and how you can use the feature to help re-engineer and modernize legacy z/OS applications.

- **Differentiating publishing services using document attribute conditioning in IBM SCORE**
  by O Michael Atogi
  The publishing services in IBM® SCORE can be conditioned using document attributes to produce varying outputs with varying styles for different compound documents for submission to different regulatory agencies. Using simple examples, this article describes how you can combine and configure these elements to create desired outputs.

- **Integrating WebSphere CloudBurst capabilities in an iPhone solution Part 2: Defining and building the iPhone application**
  by Luca Amato and Alessandro Bartoli
  This series of articles walks you through the process of creating a full client application for an Apple iPhone device that collaborates with an IBM® WebSphere® CloudBurst™ Appliance using a REST API. WebSphere CloudBurst is a new class of hardware that dispenses IBM WebSphere Application Server topologies into a cloud of virtualized hardware. Using the vast REST-based APIs provided, WebSphere CloudBurst offers many integration opportunities for a Web 2.0 environment, such as with a smartphone. Part 1 of this article series provided a high level introduction to help you prepare your development environment for creating this integrated solution. Part 2 takes the next steps toward building the sample application using the iPhone SDK, and defines the data elements that will be used for retrieving and displaying JSON data from the back end.

The Support Authority

- **Why testing in production is a common and costly technical malpractice**
  by Dr. Mahesh Rathi
  Do you know what affects the stability of your enterprise IT infrastructures? This article discusses a common characteristic that the IBM® WebSphere® Application Server SWAT team has observed while assisting clients with complex situations: either they do not have a separate test system, or the test system they do have is substantially different than their production system. If this is a characteristic that your environment shares, then you need to be aware of the destabilizing nature of this "malpractice" – and you need a plan for addressing the situation to improve stability.

Innovations within reach

- **Enhance your ESB with the REST Gateway feature in the WebSphere DataPower XC10 Appliance**
  by Charles Le Vay, Thomas Gissel and Lan Vuong
  The IBM® WebSphere® DataPower® XC10 Appliance now expands the range of clients able to access simple data grids with the release of the REST Gateway feature. Non-Java™ based clients with HTTP capabilites, including PHP and .NET clients, can utilize the XC10 appliance for elastic caching via the REST Gateway. This article provides an overview of the REST APIs and discusses how an XC10 appliance can be integrated with a WebSphere DataPower XI50 Integration Appliance as a side cache to reduce response to response time to the clients, and improve total system throughput.

The WebSphere Contrarian

- **Are you getting the most out of virtualization?**
  by Tom Alcott
  Much has happened in the area of virtualization since the topic was first discussed in this column, so this is a good time to revisit (or re-introduce) virtualization and all its flavors to help you determine which type of virtualization could benefit your organization the most, and how to make it the most effective.

Re-engineering applications using optimized local adapters on WebSphere Application Server for z/OS

Skill Level: Intermediate

James T Mulvey (jmulvey@us.ibm.com)
Senior. Software Engineer
IBM

Timothy Kaczynski (kaczyns@us.ibm.com)
Advisory Software Engineer
IBM

Frank Golazeski (fgol@us.ibm.com)
Senior Software Engineer
IBM

02 Feb 2011

IBM® WebSphere® Application Server for z/OS® V7 introduced optimized local adapters support to provide an optimized, high performance bi-directional connector for WebSphere Application Server and applications running in external z/OS address spaces. Optimized local adapters provide support for two-way synchronous local calling between applications in WebSphere Application Server and z/OS batch, CICS®, IMS™, ALCS, and USS processes. Optimized local adapters achieve optimal performance and very low latency by extending the WebSphere Application Server for z/OS local communications feature, which is based on z/OS cross memory services. It provides security context propagation in each direction with identity assertion, and it provides two-phase commit transaction support for calls between WebSphere Application Server and CICS. This article provides insight into how optimized local adapters work and how you can use the feature to help re-engineer and modernize legacy z/OS applications.

Introduction

Reusing business logic available in Enterprise JavaBeans™ (EJBs) under the IBM WebSphere Application Server runtime by applications in traditional z/OS environments has typically been challenging and costly. Although the use of XML document-based technologies for communications between applications has gained momentum over the past several years, it remains ill-suited for synchronous calls between small application units co-located on the same z/OS system. For program-to-program communications between classic application environments on z/OS and those based on Java™ and running on WebSphere Application Server for z/OS, the overhead of Java serialization and de-serialization, coupled with the conversion to and from XML (for example, for SOAP) and subsequent copying of the message to and through the TCP/IP stack, can add significant latency, which can make the cost of delays and CPU consumption excessive, and, in many cases, prohibitive.

Optimized local adapters support was introduced in WebSphere Application Server V7 as a built-in feature for WebSphere Application Server for z/OS that gives z/OS application developers and architects an optimized alternative for synchronous calling between native language programs running in legacy application environments (like z/OS batch, CICS, and IMS) and those running on WebSphere Application Server for z/OS. Optimized local adapters were designed to provide a simple way for you to quickly start integrating legacy business logic with capabilities available in Java-based programs running on WebSphere Application Server.

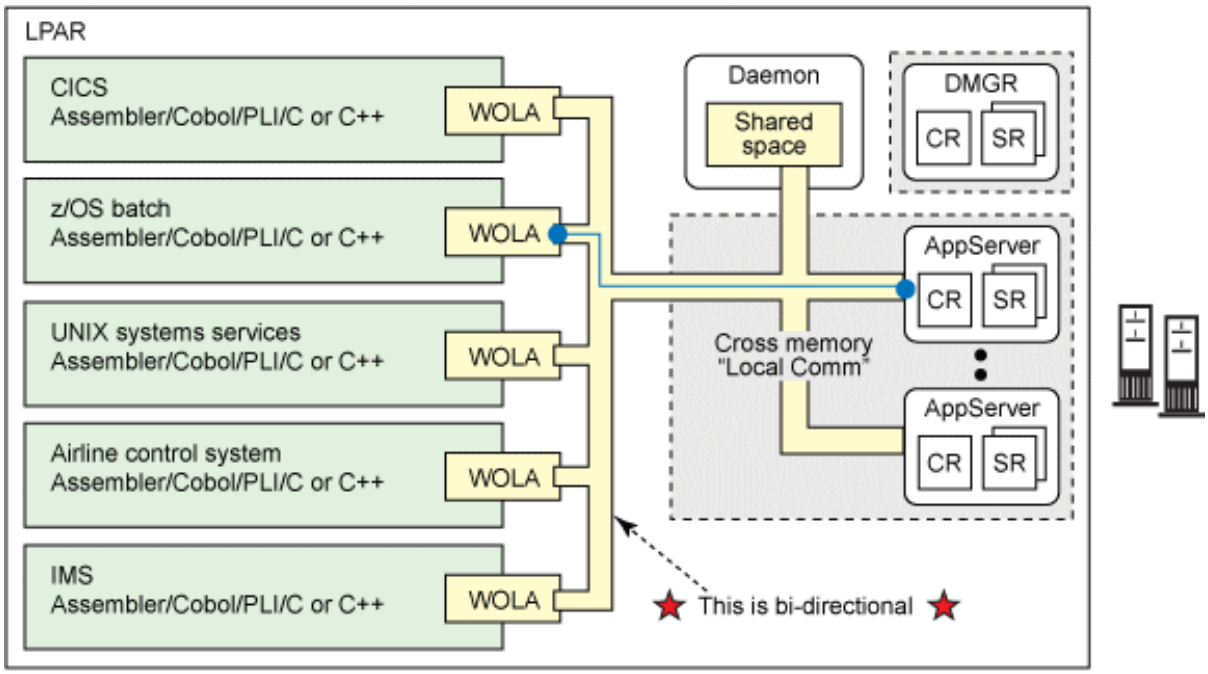
- For the COBOL, C/C++, PL/I, or Assembler language developer, the programming model is a call to an API to initiate the communications with the application server, and another API call to start invoking Java EE-based assets in WebSphere Application Server.
- For the Java developer supporting a WebSphere Application Server-based application (EJB or servlet) and calling outbound from WebSphere Application Server to a legacy application, the programming model is the familiar JCA 1.5.

Along with very low-latency bi-directional calling, optimized local adapters provide additional optimizations for essential qualities of service that make it uniquely qualified for high performance interactions between applications in WebSphere Application Server and those in external address spaces running on the same z/OS image. The transfer and assertion of the current task's user security context between address spaces is among these.

In the area of transactions, for calls between CICS and WebSphere Application Server, optimized local adapters provide local and global transaction support where CICS or WebSphere Application Server might be the initial requestor or coordinator. For calls from CICS applications to those in WebSphere Application Server, optimized local adapters provide a way to map workload context, ensuring that high priority CICS transactions retain that status once they call over to WebSphere

Application Server. This will be explained further in the sections that follow.

Figure 1. What are optimized local adapters?



This extension is implemented with a new set of modules that provide an API for programs in external address spaces to access servers using this Daemon shared space mechanism

How optimized local adapters work

Since its earliest days, WebSphere Application Server for z/OS has used a set of high performance communication services called **Local Comm**, which provides a set of cross-memory services that enable one WebSphere Application Server for z/OS server to quickly pass work to another and receive the results.

Optimized local adapters extend this technology and introduce new support that enables a standard, problem state application to switch into the needed state and drive the associated Local Comm services required to queue requests over to a specific WebSphere Application Server server. The process of introducing an address space to a specific server is called **registration**. During the optimized local adapters REGISTER API call, the application indicates which cell, node, and server it wishes to communicate with. It also provides a register name and information about the number of connections it wishes to establish and set as a maximum. Once the registration process completes, the application can begin communicating with any application installed in the target application server..

Interfaces

Calls to the optimized local adapter invocation or send request APIs do not result in an inbound JCA request to drive into the target EJB. Instead, a context switch to the WebSphere Application Server control region is made for these requests, and the request flows through a streamlined dispatch path to the target EJB method in the WebSphere Application Server servant region. For inbound calls to WebSphere Application Server EJBs, the target EJB must implement the optimized local adapter com.ibm.websphere.ola.Execute and com.ibm.websphere.ola.ExecuteHome interfaces, which include a target method, called execute(), that accepts a byte array and returns a byte array. A JAR file called ola_apis.jar is provided and contains these interfaces. This JAR file is located in the WebSphere Application Server for z/OS file system and must be included on the application's build path.

Calls from WebSphere Application Server applications (EJBs or servlets) are supported by a new resource adapter provided by WebSphere Application Server called ola.rar. External address space applications are required to first use one of the optimized local adapter server APIs (Host Service or Receive Request) and identify themselves as target services. The WebSphere Application Server application then needs to set the Register name it wishes to contact on the Connection Specification and the Service Name on the Interaction Specification before making the call into the target service.

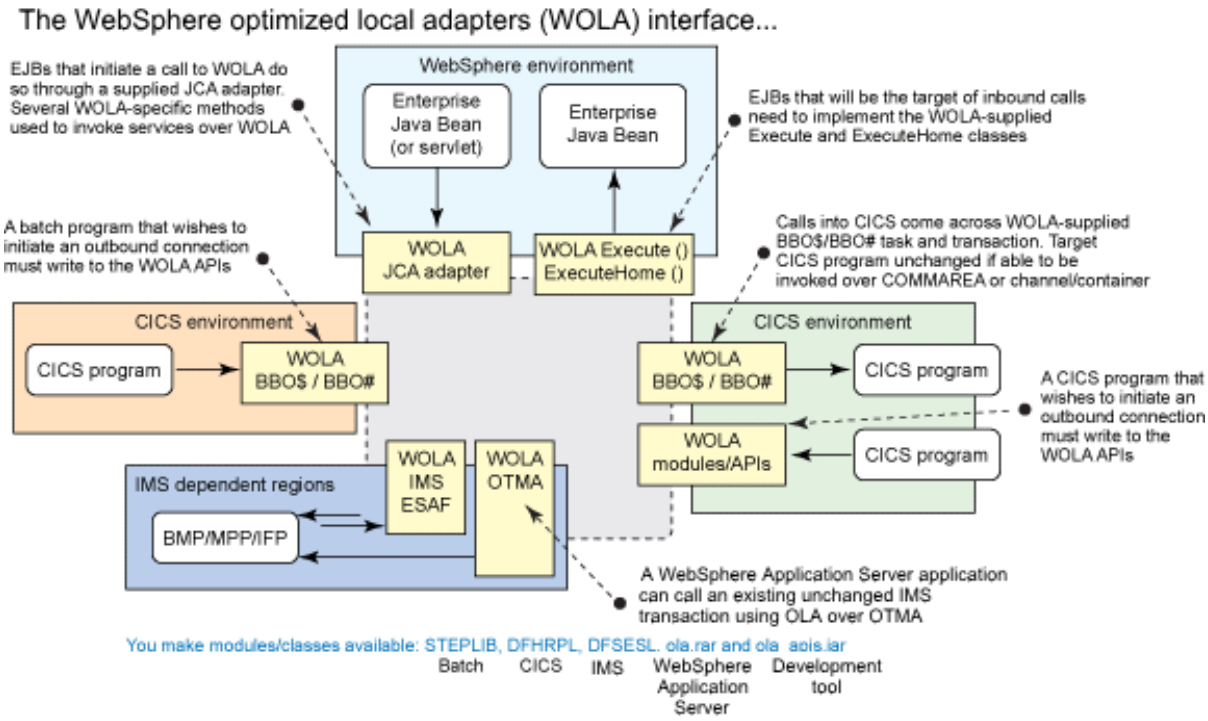
Calling unchanged CICS and IMS applications

Optimized local adapters provide capabilities that enable WebSphere Application Server applications to call existing, unchanged CICS and IMS transactions.

For CICS, the optimized local adapter CICS link server can be started in the CICS region using a new WebSphere Application Server-supplied CICS control transaction (BBOC). The link server accepts program LINK requests from WebSphere Application Server over optimized local adapters and calls the target CICS program using the EXEC CICS LINK API. It can pass a set of parameters to the program in either a COMMAREA or a CHANNEL/CONTAINER. It receives the response back and returns it to the caller in WebSphere Application Server.

For calling existing, unchanged IMS transactions, optimized local adapters provide a facility for making calls over IMS Open Transaction Manager Access (OTMA). These transactions can be driven in IMS message processing regions or fast path dependent regions.

Figure 2. The optimized local adapters interface



Mapping parameter data to and from Java objects

When calling an optimized local adapters-capable EJB, the same data structures used by your native language application can be used to generate Java classes with getter and setter methods for each attribute in the data structure. When using IBM Rational® Application Developer or IBM Rational Developer for System z®, the CICS/IMS Java Data Binding wizard in the J2C category is the wizard that creates these Java classes. The wizard takes any COBOL, PL/I, or C data structure as input and generates a Java class with the associated getter and setter methods.

In addition to mapping numeric and byte fields, it also converts string data to the appropriate codepage when the correct codepage for your native language application is selected. When receiving data from a native language application, the byte stream is fed into a new instance of the class, and getter methods are used to read the attributes. When sending data to a native language application, a new instance of the class is created and setter methods are used to populate the attributes. The class is then serialized into a byte stream which is sent to the native language application. In most cases, the Java classes generated to map data for calls using optimized local adapters can also be used with applications using CICS Transaction Gateway or IMS Connect.

Security propagation

Optimized local adapters provide support for sharing security credentials between applications in external address spaces and WebSphere Application Server applications, depending on whether the work is passed into or out of WebSphere Application Server. Let's examine security propagation in each direction separately.

Inbound to WebSphere Application Server

For batch z/OS applications making calls to WebSphere Application Server EJBs, the user ID on the running job's thread, or TCB, at the time of the Register API call is the one used for all subsequent calls to WebSphere Application Server applications. This ID is propagated and asserted in the WebSphere Application Server EJB container. There is a System Access Facility (SAF) check during the Register API call for the user's access to the target server's CBIND SAF class. If the ID making the Register API call does not have at least READ access to this class, the register call will not be accepted.

CICS and IMS to WebSphere Application Server

In order for requests sent in to WebSphere Application Server from CICS to have the user ID on the current CICS task propagated to WebSphere Application Server, the CICS startup parameters must indicate SEC=Y. Without CICS security enabled, optimized local adapters will propagate the authority of the overall CICS region instead of an individual task user.

To direct optimized local adapters to use the credentials of the particular CICS transaction that initiated the work, set the reg_flag_C2Wprop flag when calling the Register API (BBOA1REG). Because it might not be permissable for a WebSphere Application Server application to make that kind of security-based decision, WebSphere Application Server administrators have the option to either configure the server to allow or deny this behavior. To indicate that this activity is permitted, define and set the environment variable ola_cicsuser_identity_propagate to 1 in the WebSphere Application Server administrative console. (Of course, if the variable was not set and you have to create or modify it, you'll need to restart the servers within its scope in order for it to be activated. You will know the environment variable is not set if you attempt to use the Register API with the reg_flag_C2Wprop field set to 1 and the call fails with a return code 8 and reason code 21.)

For calling from IMS batch message processing regions, as for z/OS batch applications, the Job user ID is propagated and asserted in the WebSphere Application Server EJB container. For IMS message processing programs (MPPs) and fast path programs (IFPs,) the same process occurs: the identity on the current transactions thread (TCB) is propagated. For MPPs and IFPs, in order for the user ID to reflect the true IMS transaction's user ID, you must implement the IMS Build Security Environment (DFSBSEX0) exit, which tells IMS to keep the TCB user ID context in sync with the current IMS transaction's identity.

Outbound from WebSphere Application Server

For calling batch server programs from WebSphere Application Server, the user ID of the batch job is used for requests from WebSphere Application Server. There is no identity propagation in this scenario.

WebSphere Application Server to CICS and IMS

Thread level user ID propagation from WebSphere Application Server to CICS can be accomplished using the optimized local adapters CICS link server. To achieve this, you must start the link server with SEC=Y. When running in this mode, the optimized local adapter will propagate the identity of the user on the current thread to CICS and the link server will start up a link task using `EXEC CICS START TRANSID('BBO#') USER(<userid>)`. You need to ensure that the CICS SURROGAT SAF class is active and that the user ID under which the link server is running has authority to issue `START TRANSID` for the propagated user ID.

For WebSphere Application Server to IMS security propagation and assertion, you need to use the optimized local adapters over OTMA support. The WebSphere Application Server for z/OS server must be configured to run with the SyncToThread option enabled. Also, the IMS OTMA parameter OTMASE=FULL must be set. With this, the user ID under which the WebSphere Application Server application is running will be propagated and asserted in the IMS message processing or fast path dependent region where the transaction is dispatched. This does not apply to IMS batch message processing regions.

Global transactions support

(WebSphere Application Server and CICS only.)

Optimized local adapters provide two phase commit global transactions support for both CICS to WebSphere Application Server and WebSphere Application Server to CICS program invocations. Both the native language and EJB portions of an application can participate in the same global transaction using two phase commit when the native language portion of the application is running in CICS Transaction Server for z/OS.

When calling from CICS into an optimized local adapter EJB, a z/OS Resource Recovery Services (RRS) unit of recovery token is passed from CICS to WebSphere Application Server, which uses this token to create its own unit of recovery that is then cascaded to the unit of recovery received from CICS. When the CICS transaction reaches a syncpoint, RRS drives the WebSphere Application Server unit of recovery to completion.

When calling from WebSphere Application Server for z/OS into a CICS transaction

using optimized local adapters, an XA-capable transaction context is passed from WebSphere Application Server to CICS. The optimized local adapter CICS link server running in CICS reads the XA transaction context and creates a new unit of work to run the specified native language program. When WebSphere Application Server reaches a syncpoint, XA protocol messages are exchanged between WebSphere Application Server and the link server. The link server drives the appropriate functions on the CICS OTS (OT) domain to complete the unit of work.

In order to use the global transaction support when calling into an optimized local adapters EJB, you must be using CICS Transaction Server for z/OS V3.1 or higher. To use the global transaction support when calling into a CICS transaction from WebSphere Application Server, you must be using CICS Transaction Server for z/OS V4.1 or higher. Optimized local adapters decide whether or not to propagate a transaction context based on the settings of the optimized local adapter CICS link server running inside CICS, as well as the transaction attributes of the Java application running inside WebSphere Application Server.

For example, if an EJB component calls a CICS program using the optimized local adapter link server, a transaction context will only be propagated and used if the EJB is currently participating in a global transaction, and the link server was started with transaction support enabled (BBOC with TX=Y). Similarly, if a CICS program calls an optimized local adapter EJB, the transaction context will only be propagated if the EJB is deployed with a transaction attribute that supports global transactions, such as tx_required.

Workload propagation

The z/OS workload management service class, under which optimized local adapter EJB requests are dispatched, can be configured using the workload management transaction class mapping XML file. This same file is used to configure service classes for other work types dispatched in WebSphere Application Server, such as HTTP and MDB requests. Requests are mapped to a WLM transaction class based on attributes of the request, such as the URL for HTTP requests. The workload management administrator then maps transaction classes to service classes by creating definitions in workload management, which are used to assign the priority of the request.

A new request type, called **hslc**, has been added to the mapping file to support optimized local adapter requests. The service name of the request is used to map the request to a transaction class. The service name can be wildcarded, which allows groups of similar services to be easily assigned to the same transaction class.

When the native language portion of the application is running in CICS transaction Server for z/OS, the transaction name of the CICS transaction can also be used when assigning a service class to the optimized local adapter EJB request. When

the CICS transaction makes its optimized local adapter EJB request, the WLM performance block for the CICS transaction will be sent to WebSphere Application Server. The performance block will be inspected for a transaction name, and if one exists, it can be used to assign a service class for the optimized local adapter EJB request. This behavior is configured by setting a property in the workload management transaction class mapping file, and by creating definitions in workload management which map transaction names to service classes instead of transaction classes to service classes.

When using this technique for CICS transactions, it is possible to have both the CICS and WebSphere Application Server portion of the request run under the same service class by duplicating the workload management definition for the CICS transaction, and assigning the duplicate to the "CB" subsystem.

Serviceability

WebSphere Application Server for z/OS manages an internal buffer that holds optimized local adapter trace data. This 8MB buffer holds over 40,000 trace records that can be useful in diagnosing system and transaction work related to optimized local adapter requests. When the buffer fills, it wraps so that it provides a finite view back in time of optimized local adapter processing. The size of the time window depends on how quickly trace data is being created. This is affected by the amount of optimized local adapter processing, as well as the setting of the tracing granularity. While the amount of work using optimized local adapters varies, you have the ability to control how much tracing is captured.

Optimized local adapters enable you set three levels: 0 (none), 1 (coarse tracing), and 2 (detailed tracing). You might instinctively conclude that setting a more detailed trace level will generate more records and cause the buffer to wrap more quickly. This is true in general, but you can also target different trace levels for different optimized local adapter registrations, including setting a default for the entire daemon group. With different trace levels set for various registrations, the influx of data into the buffer also depends on the amount of activity in each connection being traced.

Setting up, activating, and printing trace data

Optimized local adapter trace support is dynamic, in the sense that it can be modified on the fly without requiring (for the most part) a server restart. After you decide which of the three available trace levels you wish to implement, the next thing to consider is when and where you want the tracing to take effect. There are two realms to consider. You can:

- Trace existing registrations.

- Pre-define trace parameters for registrations that have not yet been created.

Tracing for existing registrations can be set through the z/OS operator modify SETOLATRACE command. Traces can be enabled by registration name or by job name. Both of these can be used to set explicit registrations, or you can use asterisks in the names as wildcard characters to affect multiple connections in one modify command. Let's look at how the SETOLATRACE command can be used.

The operator modify command has a SETOLATRACE keyword with subsequent required parameters. The general form is:

```
F servername,SETOLATRACE=tracelevel,[ REGNAME=namestring |
JOBNAME=namestring ]
```

where :

- F is the z/OS modify console command.
- servername is the target WebSphere Application Server for the command. (This will not be a daemon name.)
- SETOLATRACE indicates you want to set the optimized local adapter trace level. The required parameter tracelevel is 0, 1, or 2.
- REGNAME indicates you want to set the trace only for registrations that are matching.
- JOBNAME indicates you likewise want to set the trace on for specific job names under which the registration was made.
- namestring specifies the registration name or the job name you want to affect. It might be handy to remember that the namestring can contain wildcard characters (asterisk is the only supported character but you can use several in the namestring. A lone asterisk matches all registrations).

Refer to the WebSphere Application Server Information Center for more details, including examples for using this command.

Be aware that you can turn off all tracing of current registrations by using either of these commands:

- `F servername,SETOLATRACE=0,JOBNAME=*`
- `F servername,SETOLATRACE=0,JOBNAME=*`

If it is not possible to enter a modify command fast enough to catch certain data in the lifecycle of a registration, you might want to predefine trace parameters so that

specific trace settings are immediately set when connections are established. You can pre-define a range of names or set a daemon group-wide default.

Server-wide tracing can be predefined using a optimized local adapter trace properties file. You can enter trace specifications in this file and then tell the daemon to activate the settings. Activating trace properties does not affect existing tracing; it affects how tracing will be set for future registrations.

To pre-define trace properties:

1. Create a file in HFS. This file should be in EBCDIC and accessible by the daemon. The file is a simple line by line specification of trace property specifications according to this syntax:
   `namestring=tracelevel`

   where `namestring` is an alphanumeric string that can contain asterisk wild characters. It specifies one or more registration names. For clarity, you can put the namestring in either single or double quotes. tracelevel is 0, 1, or 2. Any other value is considered to be 0.

   Trace property entries might look like this:

   Listing 1

```
MYREGNAME=1
"CICSREG"=2
'SHORTREG'=0
REGNAME*=1
*NAME=2
"REG*NAME"=0
*=2
```

   In the last line, *=2, sets a default trace level of 2. If no other specifications match a new registration, it gets, in this example, a trace level of 2.

   Be aware that the trace property entries are retained in the daemon's memory in the order in which they appear in the file. Therefore, when a registration is being created, its name is compared to the trace property entries from top to bottom. As soon as a match is found, the matching process ends. If, in the above example, the *=2 entry had been the first line in the file, it would match every new registration and no other entries will get a chance be compared.

   (Hint: A leading # character indicates the start of a comment, unless it is within a quoted string.)

2. Create an environment variable with this name in the administrative console:
   `WAS_DAEMON_ONLY_ola_trace_settings_file`

   This variable is a string and its value is the fully qualified path and filename of your trace properties file.

   (Big hint: Be sure to restart the WebSphere Application Server daemon to make active on all the associated nodes and servers.)

   After the daemon restart, all you need to do is activate the trace properties.

3. Activate the trace properties by issuing the modify SETOLATRACEPROPS command and the new trace settings will begin to affect future registrations. The syntax is:
   `F daemonname,SETOLATRACEPROPS`

Another way to dynamically set trace information is for the application itself to indicate a desired trace level on the call to the BBOA1REG Register API. This setting will be active for the life of the registration unless changed by a SETOLATRACE modify command. Refer to the Information Center for more details on providing the trace level using the Register API.

To direct WebSphere Application Server to print the contents of the optimized local adapter internal trace buffer to the WebSphere Application Server SYSPRINT DD, you'll need to issue this command:

```
F servername,DISPLAY,OLATRACE=jjjjj
```

where `jjjjj` = z/OS job name of external address space, or wildcard (*) for all.

You should see data that looks like Listing 2 in the SYSPRINT DD of the associated WebSphere Application Server control region:

Listing 2

```
******** OLATRACE Modify command output start ***************************
******** Trace records for Jobname: *
Trace: 2010/11/04 19:16:41.197 02 t=6C2580 c=UNK key=S2 tag= (10017001)
  Description: BBGADISP - OLA Trace record dump
  Eye Catcher:  data_address=000002200e817980, data_length=8
  +-----------------------------------------------------------------------+
  |OSet| A=000002200e817980  Length = 0008 |     EBCDIC     |     ASCII      |
  +----+-----------------------------------+----------------+----------------+
  |0000|C2C2D6C1 E3D9C1C5                  |BBOATRAE        |........        |
  +-----------------------------------------------------------------------+
  Version: 1
```

```
  Size of entry: 192
  STCK time:  data_address=000002200e81798c, data_length=8
  +-------------------------------------------------------------------------+
  |OSet| A=000002200e81798c  Length = 0008 |     EBCDIC     |     ASCII      |
  +----+-----------------------------------+----------------+----------------+
  |0000|C6CB10A9 43B3F513                  |F..z..5.        |....C...        |
  +-------------------------------------------------------------------------+
  'BBOA'+ Register name:  data_address=000002200e817994, data_length=12
  +-------------------------------------------------------------------------+
  |OSet| A=000002200e817994  Length = 000c |     EBCDIC     |     ASCII      |
  +----+-----------------------------------+----------------+----------------+
  |0000|D6D3C1C2 C3F2F6F1 40404040         |OLABC261        |........@@@@    |
  +-------------------------------------------------------------------------+
  WAS Server name:  data_address=000002200e8179a0, data_length=8
  +-------------------------------------------------------------------------+
  |OSet| A=000002200e8179a0  Length = 0008 |     EBCDIC     |     ASCII      |
  +----+-----------------------------------+----------------+----------------+
  |0000|C2C2D6E2 F0F0F140                  |BBOS001         |.......@        |
  +-------------------------------------------------------------------------+
  Job name or server name :  data_address=000002200e8179a8, data_length=8
  +-------------------------------------------------------------------------+
  |OSet| A=000002200e8179a8  Length = 0008 |     EBCDIC     |     ASCII      |
  +----+-----------------------------------+----------------+----------------+
  |0000|D6D3C1C2 C3C1D3D3                  |OLABCALL        |........        |
  +-------------------------------------------------------------------------+
  Job number: 0
  ASCB @: fc2580
  TCB @: 6c8250
  Message data:  data_address=000002200e8179c0, data_length=128
  +-------------------------------------------------------------------------+
  |OSet| A=000002200e8179c0  Length = 0080 |     EBCDIC     |     ASCII      |
  +----+-----------------------------------+----------------+----------------+
  |0000|C2C2D6C1 F0F7F0F2 C940C2C2 C7C1E2D9|BBOA0702I BBGASR|.........@......|
  |0010|E5D840E4 978481A3 85D381A3 83884040|VQ UpdateLatch  |..@...........@@|
  |0020|40404040 40404040 40404040 40404040|                |@@@@@@@@@@@@@@@@|
  |0030|40404040 40404040 40404040 40404040|                |@@@@@@@@@@@@@@@@|
  |0040|40404040 40404040 40404040 40404040|                |@@@@@@@@@@@@@@@@|
  |0050|40404040 40404040 40400000 00000000|          ......|@@@@@@@@@@......|
  |0060|00000000 00000000 00000000 00000000|................|................|
  |0070|00000000 00000000 00000000 00000000|................|................|
  +-------------------------------------------------------------------------+
******** End of OLATRACE Modify command output ***************************
```

There are two additional ways to activate tracing for optimized local adapters. These are likely to be requested by the IBM service team in support of problem determination.

- There is a WebSphere Application Server for z/OS native-level trace component for optimized local adapters that can be activated by issuing this command on the WebSphere Application Server control region :
  `F servername,TRACEDATA=G`

  After issuing this command, refer to the WebSphere Application Server control and servant region logs and look in the SYSPRINT DD.

- There is also a Java package-level trace that can be enabled using this operator command:
  `F servername,TRACEJAVA='OLA=all'`

After issuing this, the associated trace data will also appear in the WebSphere Application Server control and servant region SYSPRINT DD.

Extending the WebSphere Application Server stack to legacy z/OS applications

Optimized local adapters provide a means for z/OS applications to harness the WebSphere Application Server for z/OS stack of capabilities. For example, a COBOL or Assembler batch program could call a wrapper EJB in WebSphere Application Server, which in turn drives out to a Web service using SOAP or RESTful style calling. This eliminates the need for the batch program to include an HTTP or Web services stack in its implementation to achieve this. Similarly, with a small proxy servlet that uses optimized local adapters to call a back end server program, an existing application in batch, CICS, or IMS could be reached from callers over Web technologies such as Web services.

Conclusion

Optimized local adapters can be an integral component of an application modernization and re-engineering strategy. By introducing a simple way to perform fine-grained synchronous calling between WebSphere Application Server applications and legacy z/OS environments, optimized local adapters support adds a new dimension to what is considered possible for z/OS application architects and developers. This feature introduces the ability to create composite applications that are made up of smaller components from various application environments like CICS, IMS, and WebSphere Application Server for z/OS. It also ties together the security, transactional, and workload context components from these environments.

Resources

- Techdocs: WebSphere z/OS Optimized Local Adapters library
- IBM WebSphere Applicaton Server for z/OS Information Center (search on optimized local adapters)
- Redbook: WebSphere on z/OS - Optimized Local Adapters
- Video: WASOLA1 YouTube Channel (demonstrations)
- Washington Systems Center WOLA Education Series (Videos)
    1. The Essentials of WOLA
    2. WOLA CICS Support
    3. WOLA IMS Support
    4. WOLA Native APIs, Part 1 of 2
    5. WOLA Native APIs, Part 2 of 2
    6. WOLA Java-side Coding Considerations
- IBM developerWorks WebSphere

About the authors

James T Mulvey
Jim Mulvey is a senior software engineer in the Poughkeepsie, NY IBM labs. He is a WebSphere Application Server developer and the architect for the optimized local adapters feature. Jim's worked in WebSphere Application Server development for four years. Prior to that, he spent ten years as a developer on the IBM z/OS Language Environment and IBM Problem Determination Tools teams. Before coming to IBM, Jim worked as an independent consultant for a number of years, specializing in MVS systems and CICS applications/systems programming.

Timothy Kaczynski
Tim Kaczynski is an advisory software engineer at IBM in Poughkeepsie, NY. He has been a member of the WebSphere Application Server for z/OS development

team for ten years, with a focus on RRS, XA transaction management, resource connectivity using JCA, and optimized local adapters. He holds a Masters degree in computer science from Columbia University. In his spare time, Tim enjoys playing his viola with various orchestras in the Hudson Valley.

Frank Golazeski

Frank Golazeski is a senior software engineer at IBM. He was a WebSphere Application Server developer for the optimized local adapters feature. Frank started in WebSphere Application Server development in 2005 as part of 64-bit support for WebSphere Application Server for z/OS. He’s got a broad range of experience in IBM product development and test including millicode for enterprise servers, operating systems, e-Server subsystems, and global services development.

Differentiating publishing services using document attribute conditioning in IBM SCORE

Skill Level: Intermediate

O. Michael Atogi (atogi@us.ibm.com)
Industry Solutions Software
IBM

02 Feb 2011

The publishing services in IBM® SCORE can be conditioned using document attributes to produce varying outputs with varying styles for different compound documents for submission to different regulatory agencies. Using simple examples, this article describes how you can combine and configure these elements to create desired outputs.

Introduction

IBM Solution for Compliance in a Regulated Environment (SCORE) is a document management solution that provides end-to-end electronic document management in the Life Sciences and Healthcare industry sectors. As such, IBM SCORE supports many global industry standards, such as Electronic Common Technical Document (eCTD) for drug submissions into government regulatory agencies, Title 21 CFR Part 11, FDA guidelines on electronic records and electronic signatures, GxP, and so on.

One of the major functions provided by IBM SCORE is to combine simple documents within a compound document into a single read only PDF rendition during a workflow action. The task of generating the single PDF document from the component simple documents is called **publishing**. The workflow actions used to drive generation of the PDF rendition are send for review (SFR) and send for approval (SFA):

- In an SFR workflow, the PDF generated is sent to reviewers for annotations.

- In an SFA workflow, the PDF generated is sent to approvers who can either approve or reject the document and provide a reason and e-signature.

Starting with IBM SCORE V6.1.1.2, different document types can be published using either shallow aggregation or deep aggregation. **Shallow aggregation** uses first generation children of the compound document, while **deep aggregation** traverses the depth of the compound document structure and uses only simple documents for the aggregation. The IBM SCORE administrator can use the combination of the three publishing services described below in combination with the customizable style templates provided in IBM SCORE to generate aggregate document outputs that meet various regulatory agency requirements.

By way of simple examples, this article describes various style artifacts, how each publish services is unique, and how you can configure these publishing services using document attributes conditioning to create different desired outputs. This article assumes a basic familiarity of IBM SCORE functions and Adobe Document Descriptor XML (DDX) language.

IBM SCORE implementation

IBM SCORE is an IBM WebSphere® Portal application that is deployable to an IBM WebSphere Application Server that runs the IBM SCORE enterprise application and SCORE Web services. The SCORE enterprise application provides both a WAR module for interacting with users and EJB JAR files for database transactions. The SCORE Web services are used by remote clients to call into the SCORE application and execute actions and return the results to the caller via HTTP/SOAP requests and responses.

An IBM SCORE deployment always requires the installation of IBM WebSphere Process Server, used as the document workflow engine, IBM WebSphere Application Server Network Deployment, for administering the complex deployment environment employing different servers and services, and the WebSphere Application Server V7 Feature Pack for Web Services. IBM SCORE installation also requires a content server to securely persist and audit the voluminous data. Current implementations support IBM Content Manager, as well as EMC Documentum, each with its supporting relational databases. Other vendor Java™ EE applications are typically also deployed on various other WeSphere Application Servers to provide other miscellaneous functions to complete the solution.

Publishing services

IBM SCORE provides these **publishing services**, depending on the specific output required:

- **StructurePublishService** produces an outline of the compound document hierarchy in the same way that Windows® Explorer displays folders and files. This publishing service is of the **structure publish** type.
- **AggregatePublishService** produces an aggregate PDF rendition consisting of simple documents and nested compound documents that are all siblings of the parent compound document. This means that if there is a nested compound document that is a child of the parent, then, regardless of its depth, a PDF must be generated at the first level. If the nesting is n deep, then at each depth a nested compound document PDF must be generated, which becomes the n–i child (where i=n-1, n-2, and so on, and i <= n). The published document can also contain a table of contents, which if present starts as the first page of the aggregate PDF. If the content of a simple child document of a nested compound document changes after a nested compound document PDF has been generated, that content is not reflected in the aggregate rendition. For the content of the changed child simple document to be reflected in the nested compound document publish, a new nested compound document PDF must be regenerated using SRF, SFA, or by manual publish action. This publish service is of the **aggregate publish** type.
- **DeepAggregatePublishService** produces an aggregate PDF rendition that consists of only the simple documents within the root compound document hierarchy. The simple document renditions are used to aggregate the compound document. If during publish time a simple document does not have a rendition, then a rendition for it is requested. When all the renditions of the compound document are available, then the root compound document publish proceeds. This publish service also understands the semantics for cover page support within the document hierarchy by looking for the presence of a document subtype of _ibm_score_toc_ within the compound document; if present, any document before its position is considered cover page content. The table of contents for the published compound document is placed following the cover page (or pages, if table of contents generation has been enabled through a configuration parameter). This publish service is of the **aggregate publish** type.

Publishing style

Publishing style refers to the decoration of the document in the areas of pagination, table of contents, and footer.

Pagination controls the style used in numbering the pages of the published PDF table of contents. It can use lower Roman, upper Roman, lower Alpha, upper Alpha, decimal numerals, or none. The font, font style (normal, italic, oblique), and font size

can be controlled in the generated table of contents and on the page footers. The set of repetitious pattern of characters –- dashes, dots, lines or spaces -- that can be used to fill the space between bookmark titles and the referenced page numbers on the table of contents pages can also be customized. These customizable aspects of PDF publishing are separated into a set of templates. The templates are coded in Adobe Document Descriptor XML (DDX) language. The publishing service runs on an Adobe LiveCycle ES server, which is a DDX processor containing the Adobe Assembler service (see Resources).

IBM SCORE V6.1.1.2 ships with sample templates that contain default styles. For each default template, some of the DDX elements that can be customized are highlighted and examined. Before customizing any template, make a copy of the default template, rename it, make your updates, and then check-in the renamed copy into the repository so that it can be referenced as a configuration parameter to any of the publishing services. There are various publishing default templates but the three default templates shown below can be customized to create a different PDF appearance:

defaultTOCstyle2

The defaultTOCstyle2 template in Listing 1 controls the display of the PDF's table of contents. The bold items shown in Listing 1 are elements that are typically customized in the template:

- The **maxBookmarkLevel** attribute of the **TableOfContents** element is by default set to 3, meaning that the table of contents contains headings of level 3 or less. If you desire the table of contents to contain only up to bookmark heading level 2, then change 3 to 2.
- The **font** and **color** attributes of the **p** element are used to set the font, font size and color of the text respectively. In the default template, the font name of MyriadPro, font size of 12pt, in black color is used for bookmark heading level 1 entries in the table of contents. For bookmark heading level 2, the same font and color is used but the font size is set to 10pt. If you wish, you can use alternate syntax for specifying font family, font style, and font size, for example:
```
<p font-family="sans-serif" font-style="normal"
font-size="12pt" color="black">
```
- The **leader-pattern** attribute of the **leader** element is set to dotted by default. This means that the spaces between the bookmark title and the referenced page number on the table of contents are repetitiously filled with the dot character. If you prefer to leave those spaces blank, then replace dotted with space. Alternatively, you can specify either dashed or double-dashed for a dashed line.

- The **Center** element is used to center the table of contents on the PDF page. If a left or right style alignment is preferred, then the Left or Right element can be used respectively.

Listing 1. Excerpt of defaultTOCstyle2 template with DDX elements

```
<SCORE_DDX_TOC_TEMPLATE>

<StyleProfile name="myTOCStyle">
 <TableOfContents maxBookmarkLevel="3" createLiveLinks="true"

includeInTOC="false">
        <PageMargins right="1.5in" top="1.75in" left="1.5in" bottom="0.28in" />

<!-- ************************************************************************
 This element controls the properties of the 'first' of potentially multiple
 pages of TOC.
************************************************************************ -->
    <TableOfContentsPagePattern pages="1">
        <Header styleReference="headerForTOCPage1Style"/>
    </TableOfContentsPagePattern>

<!-- ************************************************************************
 This element controls the properties of the 'second to last' of potentially
 multiple pages of the TOC.
************************************************************************ -->
  <TableOfContentsPagePattern pages="2-last">
        <Header styleReference="headerForTOCPage2Style"/>
  </TableOfContentsPagePattern>


<!-- use this style for bookmark level 1 -->
         <TableOfContentsEntryPattern applicableLevel="1">
               <StyledText>
                      <p font="MyriadPro,12pt" color="black">
                         <_BookmarkTitle />
                         <leader leader-pattern="dotted" />
                         <_BookmarkPageCitation />
                     </p>
               </StyledText>
         </TableOfContentsEntryPattern>

<!-- use this style for bookmark level 2 -->
         <TableOfContentsEntryPattern applicableLevel="2">
                <StyledText text-indent="0.25in">
                       <p font="MyriadPro,10pt" color="black">
                            <_BookmarkTitle />
                            <leader leader-pattern="dotted" />
                            <_BookmarkPageCitation />
                       </p>
                </StyledText>
         </TableOfContentsEntryPattern>

      </TableOfContents>
</StyleProfile>

<StyleProfile name="headerForTOCPage1Style">
        <Header>
        <Center>
                <StyledText><p>Table of Contents</p></StyledText>
        </Center>
        </Header>
</StyleProfile>

<StyleProfile name="headerForTOCPage2Style">
        <Header>
```

```
            <Center>
                <StyledText><p>Table of Contents (CONTINUED)
                        </p></StyledText>
            </Center>
        </Header>
</StyleProfile>

<!-- ##### end of template #### -->
</SCORE_DDX_TOC_TEMPLATE>
```

Before committing any style change to the repository, experiment through publishing test documents and check the output to make sure that your alignment does not overlap with the existing document overlay attribute in the footer.

defaultPDFPageLabelStyle2

The **defaultPDFPageLabelStyle2** template in Listing 2 controls the display of the PDF's page number on the footer. For the most practical use of the published PDF, page numbering is necessary, and therefore the only style element that should be customized here is the page number alignment. By default, the page number is center aligned; if desired, you can leave it left or right aligned using the Left or Right element, respectively.

Listing 2. Excerpt of daultPDFPageLabelStyle2 template with DDX elements

```
<StyleProfile name="footerWithPageNumberStyle">
        <Footer>
            <Center>
                <StyledText><p><_PageLabel/></p></StyledText>
            </Center>
        </Footer>
</StyleProfile>
```

defaultSignPageLabelSyleBack

The **defaultSignPageLabelStyeBack** template in Listing 3 controls the display of the PDF's electronic signature page number on the footer on the back of the published signed PDF. Again, page numbering on the PDF is necessary for practical usage, so the only style element that should be customized here is the page number alignment. The DDX elements, attributes, and values are documented in the DDX reference in Resources.

Listing 3. Excerpt of defaultSignPageLabelStyleBack template with DDX elements

```
<StyleProfile name="footerWithPageNumberStyle">
        <Footer>
            <Center>
                <StyledText><p><_PageLabel/></p></StyledText>
            </Center>
        </Footer>
</StyleProfile>
```

ActionConfig conditioning

IBM SCORE uses XML configurations to control various processing aspects including the execution of actions. The **ActionConfig.xml** configuration file contains the inputs required for an action, and the **DocTypeConfig.xml** file contains the attributes of a document type. An attribute in IBM SCORE is an XML element defined using the Attribute element. It is metadata that describes an IBM SCORE entity or action, and it has configurable rules and a value (or values if it is a repeating attribute). For example, Listing 4 shows ActionConfig.xml with the three publishing services described above defined:

Listing 4. Sample XML definition of the publish services in ActionConfig.xml

```
<AuxService name="structurePublishService" sessionType="current" type="publish">
        <Condition name="document_type" negated="true" operator="IN">
                <Value val="clinical_study"/>
        </Condition>
        <ServicePlugin name="StructurePublishService">
                <Parm encrypt="false" name="tocStyle"
                        value="/DDX Templates/defaultTOCstyle.ddx"/>
                <Parm encrypt="false" name="formatsToIgnore" value=""/>
        </ServicePlugin>
        <Function name="publish"/>
</AuxService>

<AuxService name="aggregatePublishService" sessionType="current" type="publish">
        <Condition name="document_type" negated="false" operator="IN">
                <Value val="clinical_study" />
        </Condition>
        <ServicePlugin name="AggregatePublishService">
                <Parm encrypt="false" name="tocStyle"
                        value="/DDX Templates/defaultTOCstyle.ddx"/>
                <Parm encrypt="false" name="formatsToIgnore" value=""/>
        </ServicePlugin>
                <Function name="publish" />
</AuxService>
```

Notice that the third publish service, DeepAggregatePublishService, is missing from Listing 4. That is because prior to IBM SCORE V6.1.1.2, there were only two publishing services: StructurePublishService and AggregatePublishService. In IBM SCORE V6.1.1.2, DeepAggregatePublishService was introduced to provide functionality which the other two publishing services do not have (support for cover pages, only using simple documents rendition for the aggregation, and 1-N pagination, which was abstracted out into the style templates to make it available to the other publishing services). The full content of IBM SCORE V6.1.1.2 including a detail description of the DeepAggreatePublishService is contained in the readme document (see Resources). The presence of two publishing services that provide aggregate publishing but different characteristics requires that conditioning logic be employed to discriminate between the two services using document attributes.

In Listings 5, 6, and 7:

- The bold elements within <Condition ... > </Condition> tags within each AuxService element are condition elements that utilize document attributes.
- The bold elements within <Parm ... > </Parm> tags within the AuxService element are style-related configurations.

These parameters are inputs to the publishing service and are used to pass in the style templates [link to Publishing style section] , along with other parameters that enable or disable the option. While the parameter names are self explanatory, a complete set of parameter (Parm element) definitions are available in the IBM SCORE V6.1.1.2 readme.

Listing 5. Conditional publishing service: StructurePublishService

```
<AuxService name="structurePublishService" sessionType="current" type="publish">
        <Condition name="document_type" negated="true" operator="IN">
                <Value val="clinical_study"/>
                <Value val="pgp_doc"/>
        </Condition>
        <ServicePlugin name="StructurePublishService">
                <Parm encrypt="false" name="tocStyle"
                        value="/DDX Templates/defaultTOCstyle.ddx"/>
                <Parm encrypt="false" name="formatsToIgnore" value=""/>
        </ServicePlugin>
        <Function name="publish"/>
</AuxService>
```

Listing 6. Conditional publishing service: AggregatePublishService

```
<AuxService name="aggregatePublishService2" sessionType="current" type="publish">
        <Condition name="document_type" negated="false" operator="IN">
            <Value val="pgp_doc"/>
        </Condition>
        <Condition name="doc_subtype" negated="false" operator="IN">
                <Value val="Procedure"/>
                <Value val="Other"/>
        </Condition>
        <Condition name="document_type" negated="true" operator="IN">
                <Value val="appl_doc"/>
                <Value val="module"/>
                <Value val="submission"/>
        </Condition>
        <ServicePlugin name="AggregatePublishService">
                <Parm encrypt="false" name="formatsToIgnore" value=""/>
                <Parm encrypt="false" name="tocStyle"
                        value="/DDX Templates/defaultTOCstyle2"/>
                <Parm encrypt="false" name="includeTOC" value="true"/>
                <Parm encrypt="false" name="tocBookmarkTitle" value="_SourceTitle"/>
                <Parm encrypt="false" name="includePageLabelFooter" value="true"/>
                <Parm encrypt="false" name="pageLabelStyleLocation"
                        value="/DDX Templates/defaultPDFPageLabelStyle2"/>
        </ServicePlugin>
        <Function name="publish"/>
        <Function name="hasPendingPublishRequest"/>
</AuxService>
```

Listing 7. Conditional publishing service: DeepAggregatePublishService

```
<AuxService name="aggregatePublishService" sessionType="current" type="publish">
        <Condition negated="false" operator="OR">
                <Condition negated="false" operator="AND">
                        <Condition name="document_type" negated="false" operator="IN">
                                <Value val="pgp_doc"/>
                        </Condition>
                        <Condition name="doc_subtype" negated="false" operator="IN">
                                <Value val="Policy"/>
                                <Value val="Guideline"/>
                        </Condition>
                        <Condition name="document_type" negated="true" operator="IN">
                                <Value val="appl_doc"/>
                                <Value val="module"/>
                                <Value val="submission"/>
                        </Condition>
                </Condition>
                <Condition name="document_type" negated="false" operator="IN">
                        <Value val="clinical_study"/>
                </Condition>
        </Condition>
        <ServicePlugin name="DeepAggregatePublishService">
            <Parm encrypt="false" name="formatsToIgnore" value=""/>
            <Parm encrypt="false" name="tocStyle"
                        value="/DDX Templates/defaultTOCstyle2"/>
            <Parm encrypt="false" name="includeTOC" value="true"/>
            <Parm encrypt="false" name="tocBookmarkTitle" value="_SourceTitle"/>
            <Parm encrypt="false" name="includePageLabelFooter" value="true"/>
            <Parm encrypt="false" name="pageLabelStyleLocation"
                        value="/DDX Templates/defaultPDFPageLabelStyle2"/>
        </ServicePlugin>
        <Function name="publish"/>
        <Function name="hasPendingPublishRequest"/>
</AuxService>
```

In Listing 5, the Condition element is used to create complex logical expressions as conditions can be nested. The complete schema description of a Condition element is in the IBM SCORE XML Configuration Schema Reference. Here, it is leveraged to build a discriminating logical expression. When an action is executed to initiate a publish, the IBM SCORE application evaluates the conditions within each AuxService element of type publish to determine the first match of publish service to invoke. Therefore, each conditional expression must be written such that it excludes false positives:

- The conditions in Listing 5 say that if the compound document to be published is not of type clinical_study or pgp_doc, then use the StructurePublishService.
- Listing 6 says that if the document_type is pgp_doc and the doc_subtype is either Procedure or Other, then use the AggregatePublishService to publish the document with the style parameters declared with the Parm elements.
- Listing 7 says that if it is a clinical_study document or document_type is pgp_doc and its doc_subtype is Policy or Guideline, then use the DeepAggregatePublishService with the style parameters declared with the Parm elements.

Conclusion

Some of the publishing services and the styled elements used in IBM SCORE were described in this article. Each publishing service has unique characteristics, but the publishing styles are common and each publishing service can be configured to use any of the styled elements encapsulated in the templates to produce output that is desired (or required) by a regulatory agency. The default templates and the default configurations for each publishing service are designed to produce optimal output for that publishing service. For example, the DeepAggregatePublishService is configured to use the defaultTOCstyle2 and defaultSignPageLabelStyleBack templates by default so that they can generate 1-N pagination, including electronic signature pages. However, if any requirements for the output PDF changes, the styled elements can be customized to meet those needs. The availability of the three publishing services and the exposed styled elements through the various templates make generating various publishing outputs possible.

Resources

- Clustering of rendition and overlay servers in IBM SCORE deployment
- Assembler Service and DDX Reference for Adobe LiveCycle DDX
- IBM Solution for Compliance in a Regulated Environment (SCORE) product information
- Video: IBM developerWorks WebSphere

About the author

O. Michael Atogi
O. Michael Atogi is a software developer at the IBM Raleigh Lab in Durham, North Carolina. He has been with IBM for over seventeen years and holds an MS degree in computer science from Howard University, Washington, D.C. He currently works on IBM Solution for Compliance in a Regulated Environment (SCORE) development.

Integrating WebSphere CloudBurst capabilities in an iPhone solution, Part 2: Defining and building the iPhone application

Skill Level: Intermediate

Luca Amato (lucaamato@it.ibm.com)
Senior IT architect
IBM

Alessandro Bartoli (alessandro.bartoli@it.ibm.com)
IT Consultant
IBM

02 Feb 2011

This series of articles walks you through the process of creating a full client application for an Apple iPhone device that collaborates with an IBM® WebSphere® CloudBurst™ Appliance using a REST API. WebSphere CloudBurst is a new class of hardware that dispenses IBM WebSphere Application Server topologies into a cloud of virtualized hardware. Using the vast REST-based APIs provided, WebSphere CloudBurst offers many integration opportunities for a Web 2.0 environment, such as with a smartphone. Part 1 of this article series provided a high level introduction to help you prepare your development environment for creating this integrated solution. Part 2 takes the next steps toward building the sample application using the iPhone SDK, and defines the data elements that will be used for retrieving and displaying JSON data from the back end..

Prerequisites

Designing and creating the application is the next step in the development of a Web 2.0 application that integrates an IBM WebSphere CloudBurst Appliance with an Apple iPhone device. Before beginning, though, you should be aware of some prerequisites.

This article series assumes a general familiarity with Web2.0 technologies including Rest, ATOM, JSON, and Objective-C. The sample application described in these articles is realized using the Model-View-Controller (MVC) pattern in Objective-C using the Apple iPhone SDK 3.x, to be run on an iPhone (or iPod Touch) device with a 3.x firmware. Therefore, this development assumes some basic knowledge of the C programming language (Object-C inherits all of C's features). It is also important that you understand basic object-oriented paradigms, such as inheritance, class, interface, and so on.

> This article does not discuss the application pattern or language paradigm in detail. See Resources for additional learning resources.

Also, this exercise uses the Cocoa framework, which consists of a set of APIs, libraries, and run time code that forms the development layer for the iPhone SDK. Cocoa is implemented in Object-C and uses the MVC pattern to encapsulate views, application data, controllers and the class to mediate the management logic.

Finally, to perform the steps and run the examples from this article, you will need a computer with the Apple SDK 3.0 or higher installed, running a compatible version of Mac OS. You also need access to a WebSphere CloudBurst Appliance running in your environment.

Designing your application

The design of the sample application described here will have three windows for managing, retrieving, and displaying the data retrieved from the WebSphere CloudBurst Appliance. From left to right in Figure 1, these three application windows are:

- In the **Start window**, you enter the WebSphere ClourBurst server address and the user ID for WebSphere CloudBurst access. All access to the appliance is managed with HTTPS basic authentication using user ID and password. When the user ID and password are specified, the List window will open.
- The **List window** displays the list of the clouds managed by the appliance when you perform the WebSphere ClourBurst connection (and data retrieve). When you select a cloud from the list, the Detail window will open. (A back action is available to return in the Start window.)
- The **Detail window** displays the details of the cloud you selected. (A back action is available to return in the List window.)

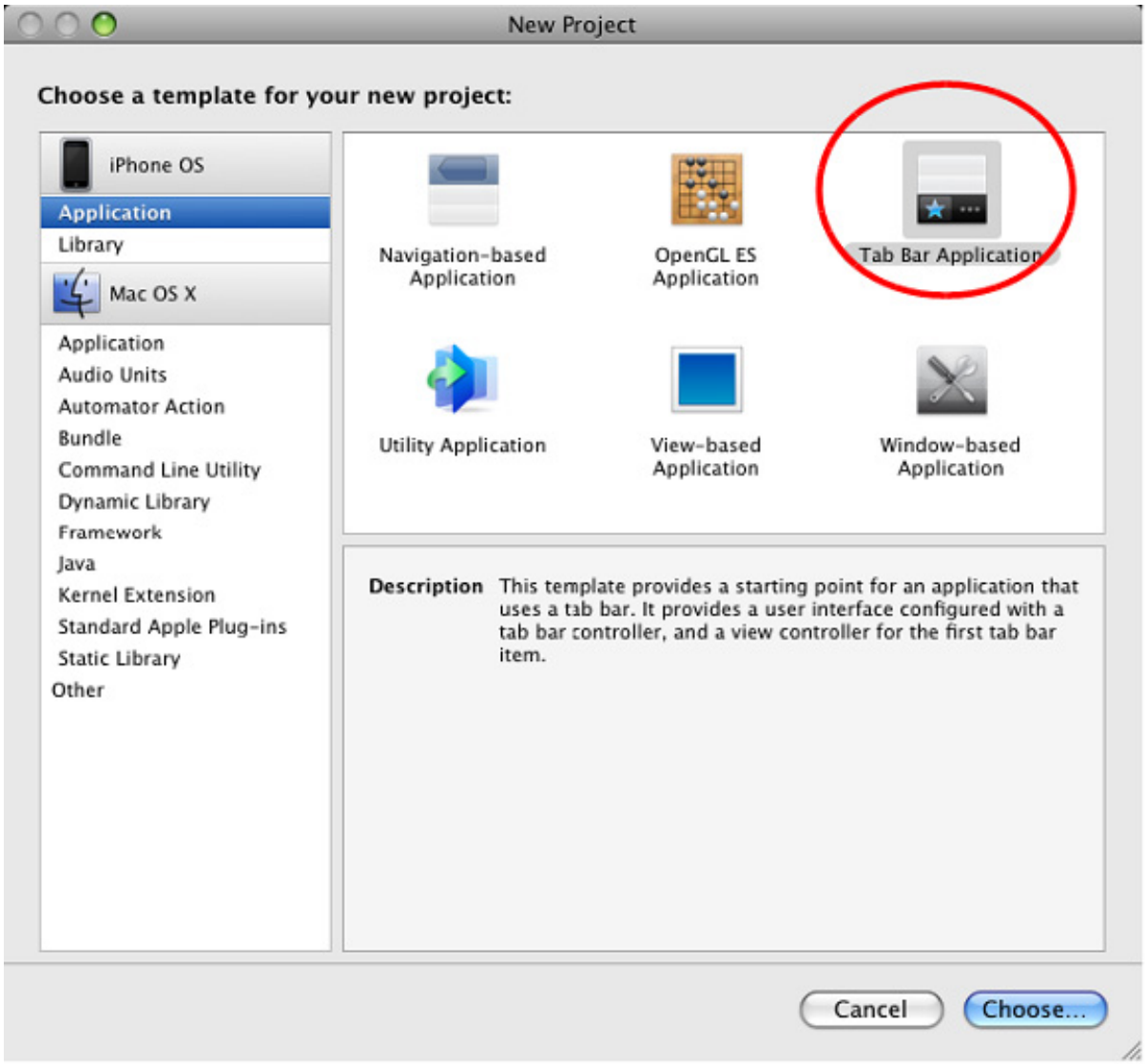
Figure 1. Application starting windows

Application template

The iPhone SDK provides several skeleton templates you can use for implementing your application (Figure 2). These are the most commonly used:

- The **View-based Application** template is for a single view application model. A simple view controller manages the primary view. With this template, you can create a simple application without any navigation. If your application requires navigation through multiple views, you can use the Navigation-based template.
- The **Navigation-based Application** template is for creating applications that traverse multiple views and require navigation between them. The navagation in the views is managed by a Views stack. You can use a Back button in any view to navagate inside your stack views.
- The **Tab Bar Application** template provides a special controller that displays a bottom navigation bar. This model is useful for an iPhone or iPod device, where a button tab provides a series of shortcuts to reach application functions.

Figure 2. Available SDK application templates



For the sample application described here, you will use the Tab Bar Application template. This template creates a tab navigation bar with two associated views. The Start window will be created as the First View. You can use the second view to implement other functionality.

Create the project

As described in Part 1 of this series, you now need to create a new project:.

1. Lauch **Xcode** from your iPhone SDK (located in /Developer/Application by default).
2. In the Product popup menu, make sure **iPhone** is selected, then select

File > New Project. Click the **Tab Bar Application** icon (Figure 2). This template provides a starting point for your application.

3. Click **Choose**.

4. Insert your project name, such as `WCA001`, and then click **Save**.

5. Select **Build > Build and Go**.

6. The tool will compile and install your application in the iPhone Simulator. The iPhone simulator starts your application automatically. You should see a panel similar to that shown in Figure 3.

Figure 3. First View

With your skeleton now avalable, you are ready to create your application.

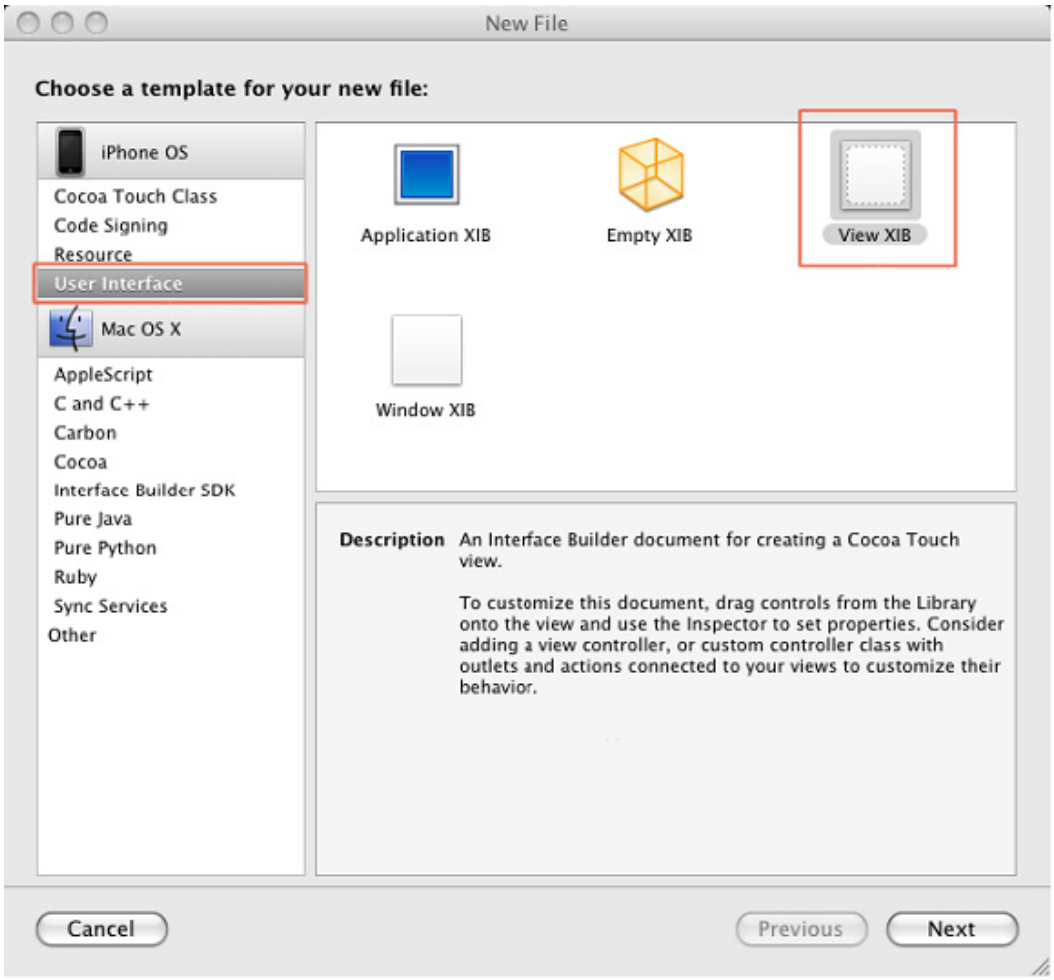
Create a working window

As mentioned earlier, your application is built using an MVC paradigm. When you select an application template, the tool automatically generates an application delegate (**WCA000AppDelegate.m**) and dummy windows (**MainWindows.xib** and **SecondWindow.xib**). You can find these windows inside the NIB Files groups. For each window, you will create a view and associated controller using this naming convention: `xxxView` and `xxxViewController`. Therefore, you can call the one window view **OneView** and its controller **OneViewController**.

Create your first view

1. In the iPhone SDK, select **File > New File**.
2. Select **User Interface** and then the **View XIB** icon (Figure 4).

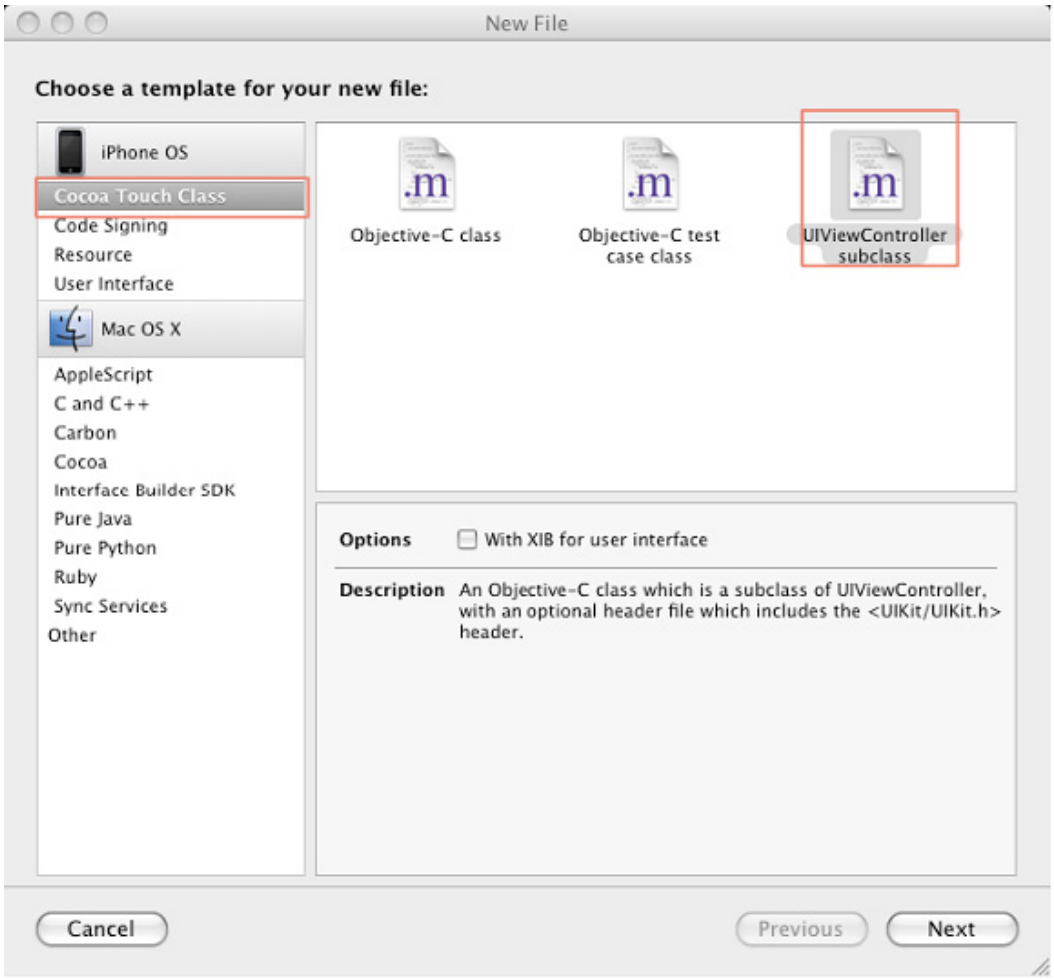
Figure 4. Create View



3. Click **Next**.
4. Insert the NIB file name, `OneView`.
5. Click **Finish**.

Create your first view controller

1. Select **File > New File**.
2. Select **Cocoa Touch Class** and the **UIViewController subclass** icon (Figure 5).
   Figure 5. Create view controller class



3. Leave **With XIB for user interface** unchecked. Click **Next**
4. Enter the view controller file name, `OneViewControler`. Be sure that

the **Also create "xxx.h"** is checked; you will need to create the **myFile.m** and the **myFile.h** files

5. Click **Finish**.

As a result of the above steps, you will now find your new class files **OneViewControler.h** and its **OneViewControler.h** in your file repository.

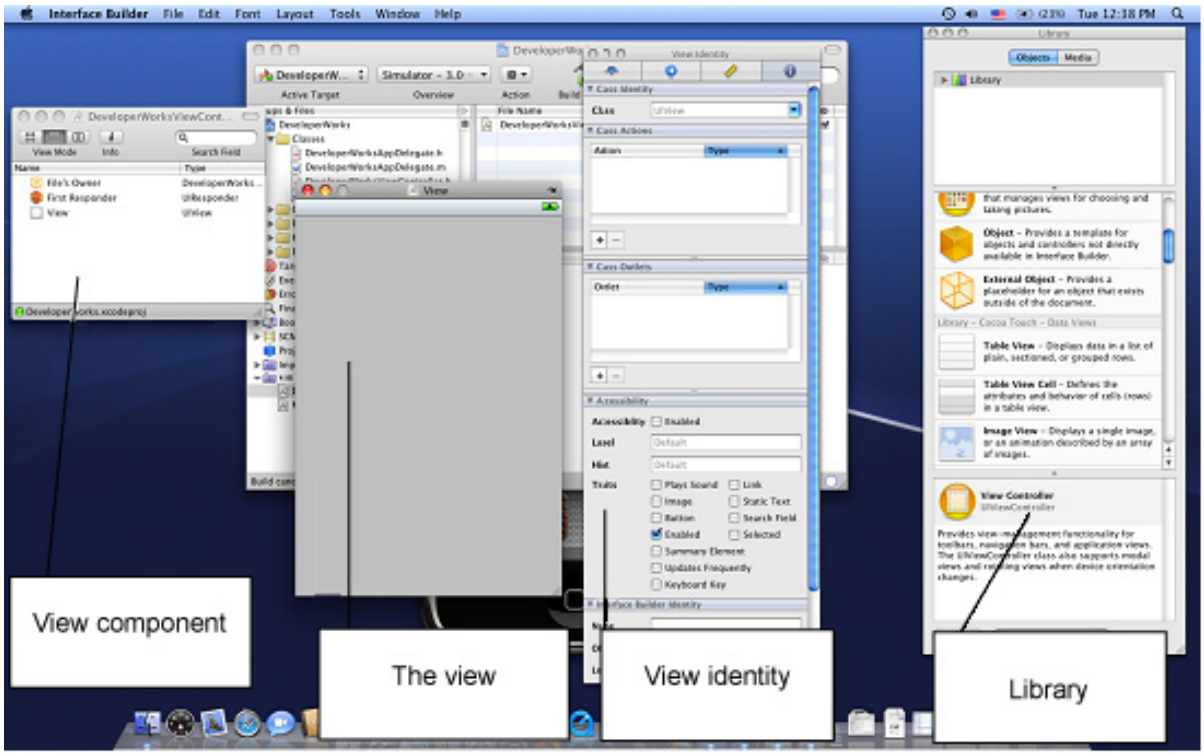
Connect the view to its controller

In the MVC paradigm, each view has a controller class. When an event happens in the view, the Cocoa framework will call the controller. You will enter all the logic to manage the event inside the controller. You must define the controller for any defined windows.

Double-click on the **OneView.xib** view file to open the Interface Builder (Figure 6). The Interface Builder is composed of four panels:

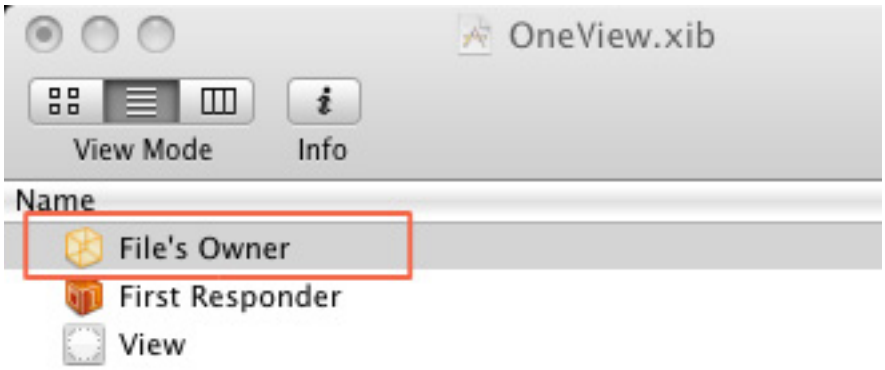
- The **View** contains the displayed object.
- **View identity** contain the view characteristic.
- **Library** is a collection of Cocoa objects that you can use to improve the view.
- **View component** handles the View component and the connection with its controller.

Figure 6. Interface Builder desktop

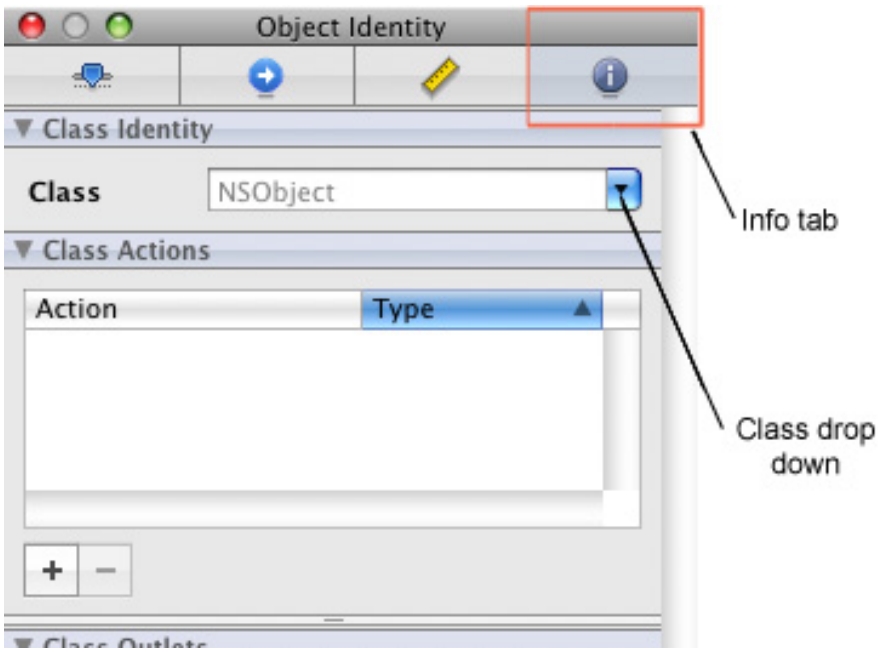


1. Click on the View.

2. Select **File's Owner** element in View Component panel (Figure 7).
Figure 7. file's Owner in View Component

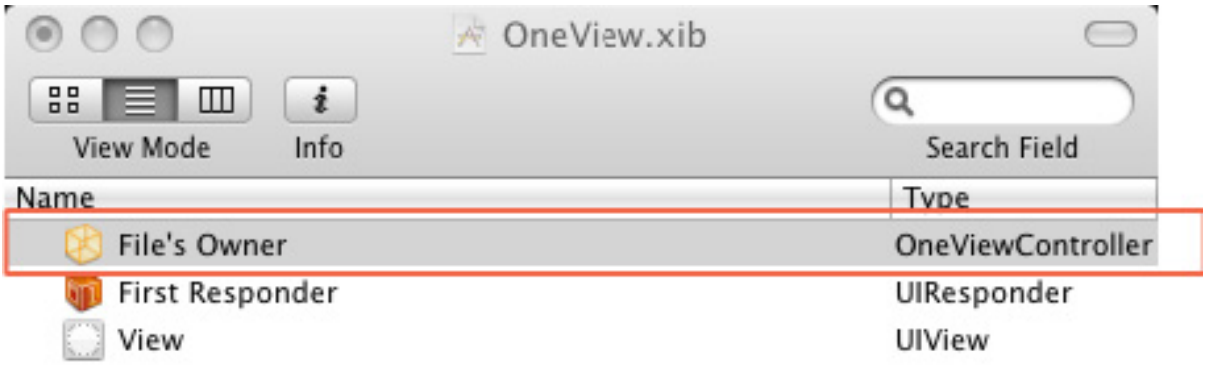


3. In the View Identity panel, select the **Info** tab (Figure 8). In the Class drop down list, select your view controller class, **OneViewController**.
Figure 8. Class drop down list



At the end, you can see the view controller class manager of your View in the View Component (Figure 9).

Figure 9. The View Controller of the selected View



Realize the views

As described above, you must create a View NIB file and view controller for each application window to associate any view to the correct view controller class. To do this, you must to repeat the create working windows steps for each application component view. The sample solution described here has three windows. Table 1 shows each window's name and its view controller class.

Table 1. Application windows

Window name	View NIB file (*.xib)	View Controller class (*.m,*.h)
Start window	OneView	OneViewController

List window	ListElementView	ListElementController
Detail window	DetailSystemView	DetailSystemViewController

Create Start window

Next, you must add the Cocoa view object on the view. The Interface Builder in the SDK makes this wasy. When you open a view (for example, the OneView.xib file) by double-clicking, the SDK opens the Interface Builder.

You are now ready to load your background bitmap image into the project. PNG files are the best image type to use for this purpose, but you can use any JPG image as an alternative. Be attentive to the image size. This example uses the image file WCALogon.png (320x367 pixels). The best way to organize images inside your project is to create a specific group for all your images:

1. Right click on **Group & Files**.
2. Select **Add > New Group**.
3. Set the name of your group to `Images`.
4. Right click on the **Images** group.
5. Select **Add > Existing Files..**.
6. Navigate to the image you want to add to your view, then click **Add**.
7. Be sure to check **Copy items into destination group's folder**.

Your image is now imported in the project group. Remember you need to do the same for all the images or other files that you need in your project. Now would be a good time to do this.

The Start window is made up of these components:

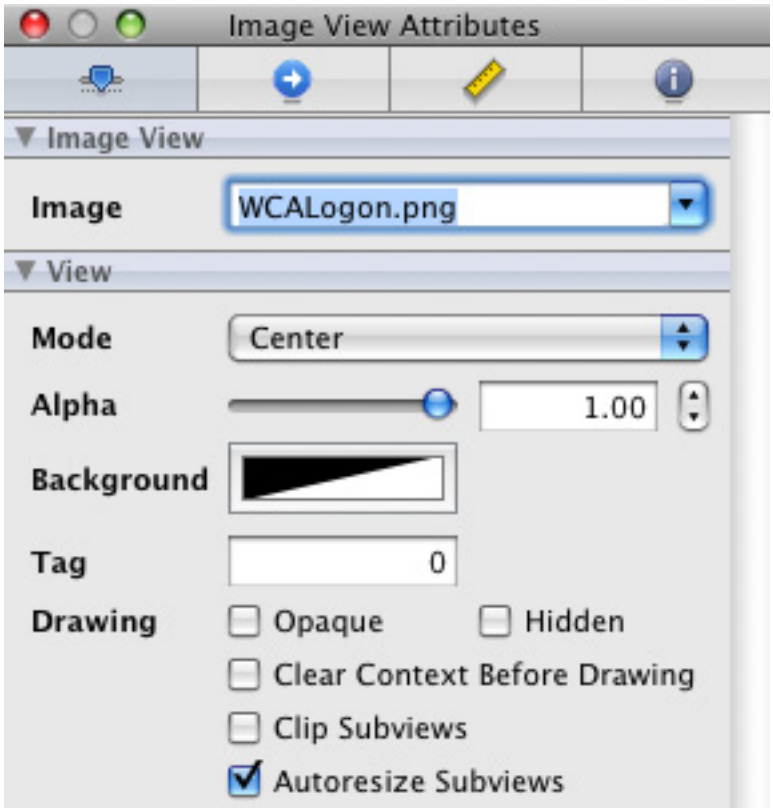
- **Image view:** your background bitmap (UIImageView class).
- **Server text field:** the IP address input field (UITextField class).
- **Userid text field:** the administrator WebSphere CloudBurst ID field (UITextField class).
- **Load button:** the button to lanch the WebSphere CloudBurst data retrieved action (UIButton class).
- **Test button:** used for testing the application in "unplugged" mode

(UIButton class).

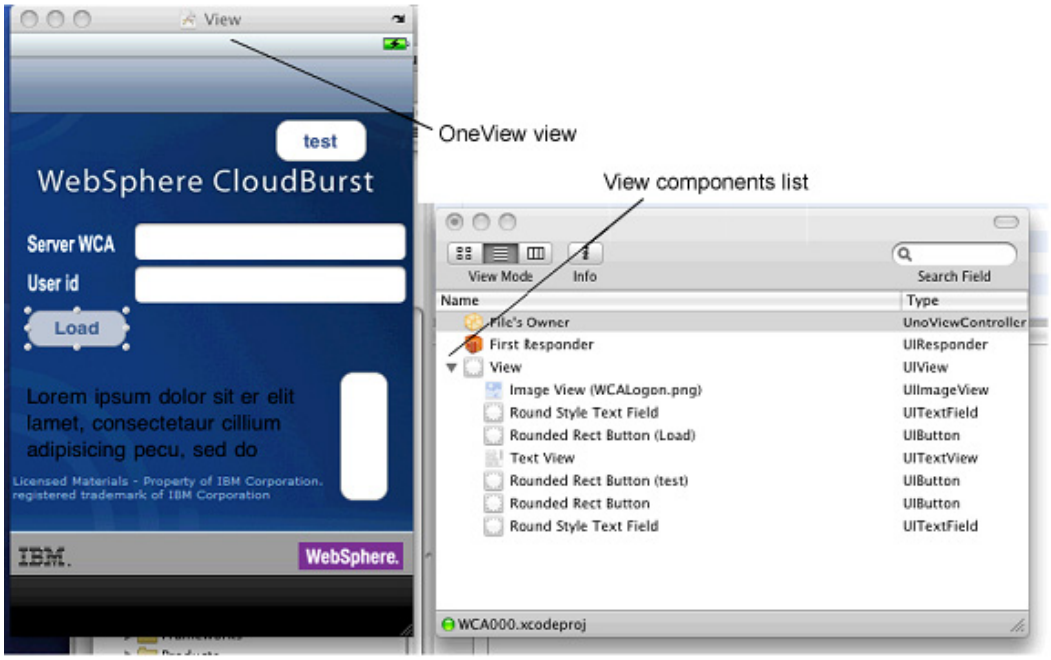
- **Next button:** used to lanch the next action (UIButton class).

The display components are present in the Library windows. You can drag and drop the necessary items into your view in the appropriate locations. The editor helps you align components inside the view. The list and order of your View components is displayed in the View component.

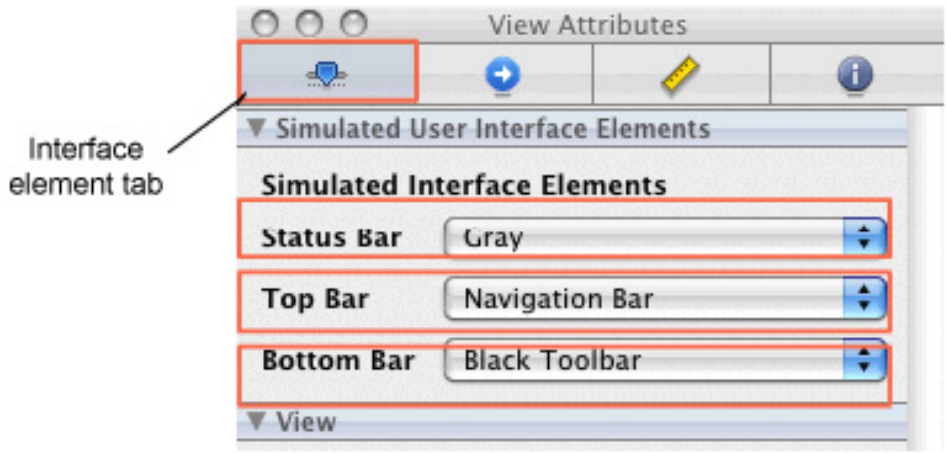
8. To assign a specific value (tag, image, and so on) to your componet, you can use the View Identity to set the image assigned to your UIImageView object. Select your object on the view. In the View Identity Attrubutes, select the PNG file you wish to assign to the view object (Figure 10).
Figure 10. UIImageView Identity Attributes



9. In the same window, you can set other image attributes as you wish. Figure 11 shows the view and its components list (Component view).
Figure 11. The OneView structure



10. Before you save OneWindow, check to make sure the Simulated User Interface Element contains the correct layout. In the View Identity, be sure these values are set as shown in Figure 12.
Figure 12. OneView layout

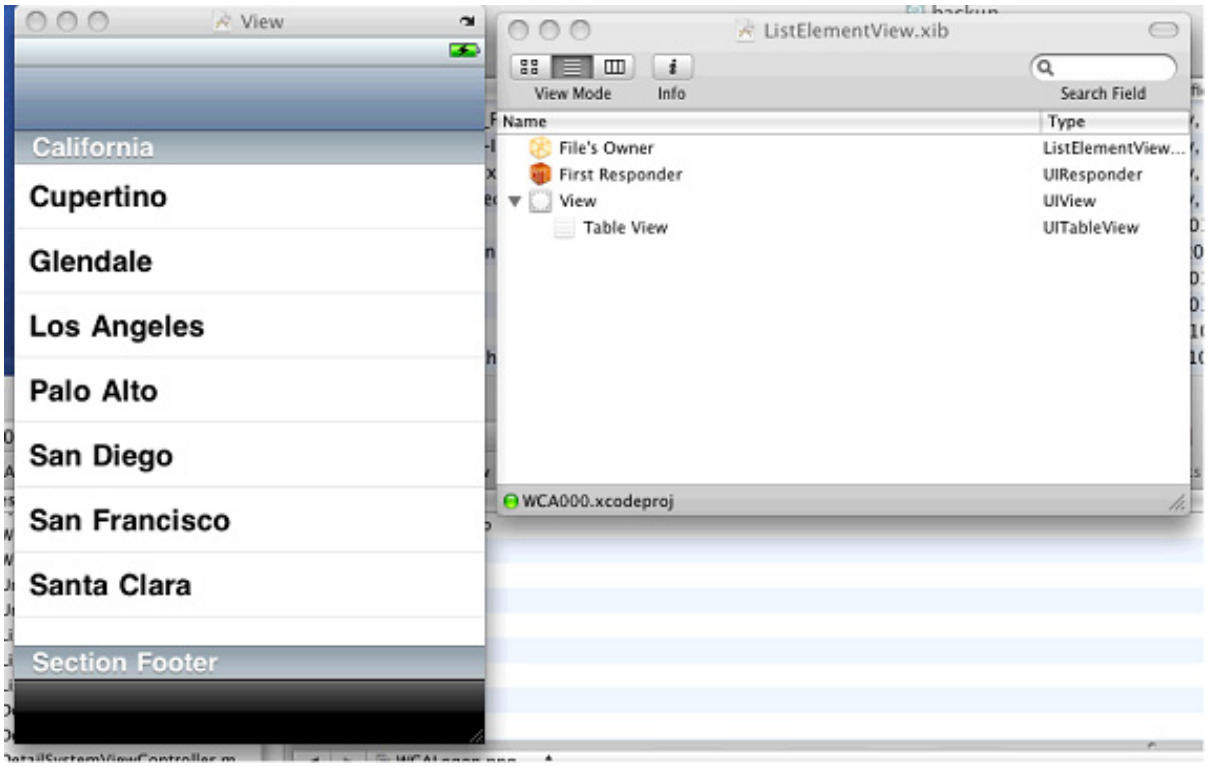


You can now save your view by selecting **File > Save**.

Create List window

The List Window is made up of a Table view, which is your cloud list (UITableView class), as shown in Figure 13.

Figure 13. ListElementView structure



Insert the view component in the same manner as you did for the Start window.

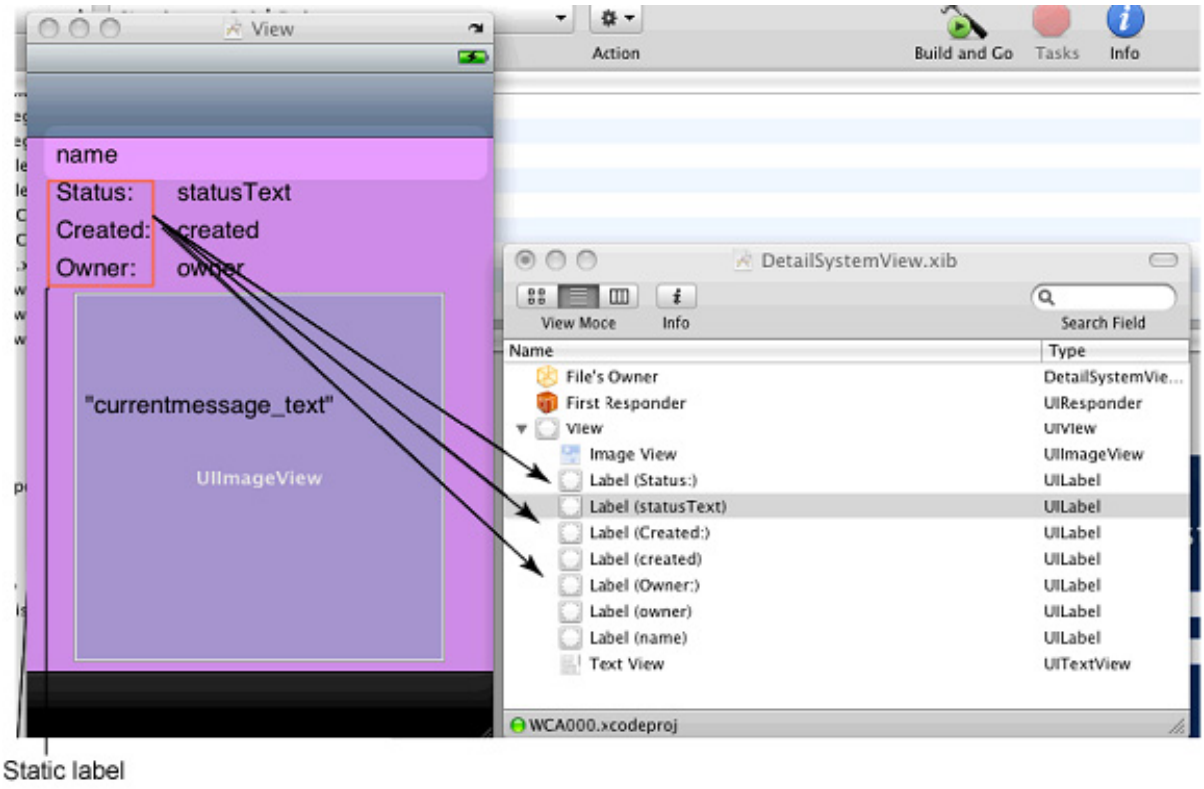
Create Detail window

The Detail window is made up of these components:

- **Image View:** shows the cloud status (UIImageView class).
- **Label:** static text, Status: (UILabel class).
- **Label:** status string description, text retrieved from WebSphere CloudBurst (UILabel class).
- **Label:** static text Created: (UILabel class).
- **Label:** created string description text, retrieved from WebSphere CloudBurst (UILabel class).
- **Label:** a static text Owner: (UILabel class).
- **Label:** owner string description text, retrieved from WebSphere CloudBurst (UILabel class).
- **Label:** name string description text, retrieved from WebSphere CloudBurst (UILabel class).
- **Text View:** currentmessage_text string description text, retrieved from

WebSphere CloudBurst (UITextView class).

Figure 14. DetailSystemView structure



Remember to assign the correct value to the static label using Identity windows in the same manner as you assigned the PNG file for the backgroung image of OneView.

Define class for controller

When you created the NIB file, you also created the controller class for each view. Each view controller receives the action every time the user performs an action on the view. The framework creates the class skeleton automatically, but you must complete the class with your code. Specifically, you need to create an instance variable for each dynamic component that you defined on the view. You also need to create an interface to intercept the view action to transfer information when you navigate between the views.

In the MVC framework, you define one view controller for each view. You can also create a class to manage the data object, which will contain all the information that you retrieved from the JSON file. (Details on the JSON to data transformation are beyond the scope of this article. Several open source JSON parsers are available.

You can use the parser that best meets your requirements and include it in your development process.) For the purpose of this exercise, this statement is inserted in the class to indicate where the JSON parser would be included (this line will be different depending on your parser):

```
#import "JSON/JSON.h"
```

Also for the purpose of this article, Cocoa framework defintions are used as is (IBOutlet). (Details on the Cocoa framework are beyond the scope of this article. For more information, see Resources.)

OneView controller in test mode

In the OneViewController class, you defined an object to test the application in "unplugged" mode. You will remove this when you insert the code that manages the HTTP syndication. This is the only this class that will be modified; because the others do not perform any network communication, they will not change.

First, insert the IBOutlet definition, for managing the view component into the OneViewController.h file (Listing 1). For the action, you will use the -(IBAction) definition. Also, some constant values are defined to improve the readability of the code.

Listing 1. OneViewController.h

```
//
//  OneViewController.h
//  WCA000
//
//  Created by Luca Amato on 9/25/10.
//

#import <UIKit/UIKit.h>
#import <Foundation/Foundation.h>

#define FILENAMEARCH @"archive"
#define PROTOCOL @"https://"

#define SERVERPSW @"wca_admin_psw"
#define API @"/resources/virtualSystems"
#define USER @"wca_admin_id"

@interface OneViewController : UIViewController {

        IBOutlet UITextField *field1 ;
        IBOutlet UITextField *field2 ;
        IBOutlet UITextView *description;

        NSMutableData *receivedData;

        NSString *inputString;

        NSMutableData *responseData;
}

@property (nonatomic, retain) UITextField *field1 ;
@property (nonatomic, retain) UITextField *field2 ;
```

```
@property(nonatomic, retain) NSString *inputString;
@property (nonatomic, retain) IBOutlet UITextView *description;
@property (nonatomic, retain) NSMutableData *receivedData;

-(IBAction) buttonPressedTest;
-(IBAction) buttonDownloadImage;
-(IBAction) buttonNext;

@end
```

You can now save the file by selecting **File > Save**.

ListElementView controller

Add the definitions for managing the view component into the ListElementView.h file (Listing 2), and then save the file.

Listing 2. ListElementViewController.h

```
//
//  ListElementViewController.h
//  WCA000
//
//  Created by Luca Amato on 9/25/10.
//

#import <UIKit/UIKit.h>

#define MAINLABEL_TAG 1
#define SECONDLABEL_TAG 2
#define PHOTO_TAG 3

@interface ListElementViewController : UIViewController {
        NSMutableArray *listElement;
        NSString *descriptionText;
}

@property (nonatomic, retain) NSMutableArray *listElement;

-(IBAction) loadJsonText:(NSString *)str;
- (void)setDescpText:(NSString *)_text;
@end
```

DetailSystemView controller

Add the definitions shown in Listing 3 to the DetailSystemView.h file (Listing 3), and then save the file.

Listing 3. DetailSystemViewController.h

```
//
//  DetailSystemViewController.h
//  WCA000
//
//  Created by luca amato on 9/27/10.
//

#import <UIKit/UIKit.h>
#import "VirtualSystems.h"
```

```
@interface DetailSystemViewController : UIViewController {
        IBOutlet UILabel *name;
        IBOutlet UILabel *currentStatusText;
        IBOutlet UILabel *created;
        IBOutlet UILabel *owner;

        IBOutlet UIImageView *statusImg;

        IBOutlet UITextView *currentMessageText;

        VirtualSystems *aSystem;
}

@property (nonatomic, retain) VirtualSystems *aSystem;

- (void)setVirtulaSystem:(VirtualSystems *)system;

@end
```

Define data object

Your application needs to manage a cloud list. The application will retrieve this list from WebSphere CloudBurst (in JSON fomat). As discussed in Part 1, the conversion from JSON format to Object-C data is performed by a JSON open source parser. Typically, this code obtains a Dictionary (NSMutableDictionary class) object. You should create a class that contains an instance of the object that contains the data from the parser. The VirtualSystems class (Listings 4 and 5) implements methods to return all the necessary values, wrapping the dictionary object inside. Regarding behaviours, you need to define only the get message. The object will be created by using this constructor:

```
-(id)initWithJSON:(NSDictionary *)jsonDict name:(NSString *)
n;
```

Table 2 lists the VirtualSystems class behaviours.

Table 2. VirtualSystems behaviours

Behaviours	JSON tag	Notes
getName	@"name"	name associated with this virtual system
getId	@"id"	the ID of the virtual system
getCurrentStatus	@"getcurrentstatus"	Specifies a string representing the current status of the virtual system
getCurrentStatusText	@"getcurrentstatus_text"	Specifies the textual representation of currentstatus
getOwner	@"owner"	Specifies the URI of the user that owns this virtual system
getCurrentmessage	@"currentmessage_text"	Specifies the textual

		representation of currentmessage
getCreated	@"created"	Specifies the creation time of the virtual system

Listing 4. VirtualSystems.h

```
//
//  VirtualSystems.h
//  WCA000
//
//  Created by luca amato on 9/27/10.
//

#import <UIKit/UIKit.h>

@interface VirtualSystems : NSObject {
        NSString *name;
        NSDictionary *element;
}

@property (nonatomic, retain) NSDictionary *element;
@property (nonatomic, retain) NSString *name;

-(id)initWithJSON:(NSDictionary *)jsonDict name:(NSString *)n;

@end
```

Listing 5. VirtualSystems.m

```
//
//  VirtualSystems.m
//  WCA000
//
//  Created by luca amato on 9/27/10.
//

#import "VirtualSystems.h"

@implementation VirtualSystems

@synthesize element, name;

-(id)initWithJSON:(NSDictionary *)jsonDict name:(NSString *)n {
NSLog(@"VirtualSystems.jsonString:%@",jsonDict);
        self.name = n;
        self.element = jsonDict;
        return self;
}

-(NSObject *)getName {
        NSObject *value=[element objectForKey:@"name"];
        return value;
}

-(NSInteger *)getId {
        NSDecimalNumber *id=[element objectForKey:@"id"];
        NSInteger intId = [id intValue];
        return intId;
}

-(NSString *)getCurrentstatus {
        NSString *status=[element objectForKey:@"currentstatus"];
```

```
        return status;
}

-(NSString *)getCurrentstatusText {
        NSString *status=[element objectForKey:@"currentstatus_text"];
        return status;
}

-(NSString *)getOwner {
        NSString *theOwner=[element objectForKey:@"owner"];
        return theOwner;
}

-(NSString *)getCurrentmessage {
        NSString *theCurrentmessage=[element objectForKey:@"currentmessage_text"];
        return theCurrentmessage;
}

-(NSString *)getCreated {
        NSDecimalNumber *theCreated=[element objectForKey:@"created"];
        NSLocale *usLocale = [[NSLocale alloc] initWithLocaleIdentifier:@"en_US"];
        NSString *decimalString = [ theCreated stringValue];
        [usLocale release];
        return decimalString;
}

@end
```

Test file for unplugged mode

To test the application in uplugged mode, you use a JSON static file stored inside the project. You can use the file shown in Listing 5 as an example, or you can download a JSON file directly from your WebSphere CloudBurst Appliance. (To store it inside the project, use the same command that you used to upload an image into the project.)

Listing 6. example.sql file

```
[
   {
      "desiredstatus_text": null,
      "currentstatus_text": "Arrestato",
      "created": 1260941860895,
      "name": "Single Server",
      "currentstatus": "RM01011",
      "desiredstatus": null,
      "currentmessage": "RM07153",
      "pattern": "/resources/patterns/1",
      "owner": "/resources/users/6",
      "updated": 1260943821497,
      "id": 51,
      "currentmessage_text": "Stopped"
   },
   {
      "desiredstatus_text": null,
      "currentstatus_text": "Avvio",
      "created": 1263868560454,
      "name": "05WASadmin Cloned Lab Virtual System",
      "currentstatus": "RM01005",
      "desiredstatus": "",
      "currentmessage": "RM07054(\"05WASAdmin+Lab+Application\")",
```

```
      "pattern": "/resources/patterns/43",
      "owner": "/resources/users/15",
      "updated": 1263869039755,
      "id": 59,
      "currentmessage_text": "Running"
   },
   {
      "desiredstatus_text": null,
      "currentstatus_text": "Avviato",
      "created": 1263862881511,
      "name": "LucaV",
      "currentstatus": "RM01006",
      "desiredstatus": "",
      "currentmessage": "RM07045",
      "pattern": "/resources/patterns/1",
      "owner": "/resources/users/4",
      "updated": 1263863256864,
      "id": 56,
      "currentmessage_text": "Ready for use"
   }
```


Save and test

Save all your files. Test your code by selecting **Build > Build and Go** to compile and run the application. If errors occur, make any necessary modifications and then recompile the project and re-test.

Conclusion

The second installment of this series of articles, describing how to integrate an iPhone application with WebSphere CloudBurst capabilities to access and display back end data, described how to prepare the windows for your sample application and how to create the class to manage the data object. Sample code is included with this article for comparison to supplement your testing. In Part 3, you will see how to manage the data flow and pass the data through the controller to appear in the actual application windows.

Downloads

Description	Name	Size	Download method
Sample code	1102_amato2_WCAsample.zip	1.5MB	HTTP

Information about download methods

Resources

- IBM WebSphere Cloudburst Appliance product information
- Cloud computing zone
- Series: Managing your private cloud
- Series: Cloud computing for the enterprise
- Enable C++ applications for Web service using XML-RPC
- Apple Developer resources
- Mac OS X Cocoa Frameworks
- iOS Dev Center
- JSON
- Objective-C tutorial
- IBM developerWorks WebSphere

About the authors

Luca Amato
Luca Amato is an IBM certified architect who has more than 15 years experience in software development. He has participated as a consultant and project leader in many projects based on web technology. Luca holds a degree in Information Technology from the University of Pisa, Italy, and teaches a language theory coursy at the University of Pavia for the Informatic department. Luca is an integration solution architect for WebSphere at IBM Italy in Milan, Italy. He works in the WebSphere group with many years of experience in SOA-based solutions. He is also a member of the OpenSource community in IBM.

Alessandro Bartoli
Alessandro Bartoli is an IBM consultant with several years of experience in Software Development. He has worked as a consultant on some IT Governance Management projects and has four certifications in ITIL v3 (Information Tecnology Infrastructure Library). In the past, he also worked as an IT Architect for WebSphere Application Server, WebSphere Portal, and WebSphere Process Server environments. Alessandro holds a Master's degree in Information Technology Engineering from the Politecnico of Milano.

The Support Authority: Why testing in production is a common and costly technical malpractice

Skill Level: Introductory

Dr. Mahesh Rathi
WebSphere Application Server SWAT Team
IBM

02 Feb 2011

Do you know what affects the stability of your enterprise IT infrastructures? This article discusses a common characteristic that the IBM® WebSphere® Application Server SWAT team has observed while assisting clients with complex situations: either they do not have a separate test system, or the test system they do have is substantially different than their production system. If this is a characteristic that your environment shares, then you need to be aware of the destabilizing nature of this "malpractice" – and you need a plan for addressing the situation to improve stability.

In each column, The Support Authority discusses resources, tools, and other elements of IBM® Technical Support that are available for WebSphere® products, plus techniques and new ideas that can further enhance your IBM support experience.

This just in...

As always, we begin with some new items of interest for the WebSphere community at large:

- Are you ready for Impact 2011? Join us for Impact 2011, April 10-15, 2011 in Las Vegas Nevada, at The Venetian and The Palazzo Hotels. Register before February 18 and receive an Early Bird discount. And check out these top 5 reasons to attend Impact (PDF, 115KB), the one conference where business and IT leaders can explore together how to achieve greater business agility.

- Earlier this year, the IBM Support Portal was named one of the Top Ten Support Sites of 2010 by the Association of Support Professionals. Have you tried the IBM Support Portal yet? All IBM software products are now included, and all software product support pages have been replaced by IBM Support Portal. See the Support Authority's Introduction to the new IBM Support Portal for details.
- Check out the IBM Conferences & Events page for a list of upcoming conferences
- Learn, share, and network at the IBM Electronic Support Community blog on developerWorks.
- Check out the new Global WebSphere Community at websphereusergroup.org. Customize the content on your personalized GWC page and connect to other "WebSpherians" with the same interests.
- Several exciting webcasts are planned in February at the WebSphere Technical Exchange. Check the site for details and become a fan on Facebook!
- **IBM Support Assistant 4.1.2 is now available.** IBM Support Assistant 4.1.2 delivers several defect fixes and a new version of its quick data collection tool, ISA Lite. These new features are now available in ISA Lite:
  - Once your inventory is collected in ISA Lite, you can easily view the inventory in a browser.
  - ISA Lite uses Ant 1.8, leveraging the latest technology available.
  - You can more easily view the menu options because each menu option running in console mode has a number associated with it.
  - You can pause the processing of a response file in different scenarios:
    - Pause the response processing for a defined period (for example, to collect trace).
    - Pause the response processing until a console response is received (for example, to enable a problem to be recreated, which is a step in a lot of plug-in scripts).
  - ISA Lite now has a more extensible format, using name-value pairs for the response file, which enable:
    - Users to add comments inside the response file.
    - Response files to contain name-value pairs that cover all the relevant questions.

    - Users to easily edit response files by hand for customizing for different systems.
- ISA Lite version information is automatically written to the console and log at startup; the same information can be found under the Help menu, and by passing -version to the start scripts.
- You can run ISA Lite with the -help option to view information on how to use the tool.
- You can select an alternate file transfer option when the selected file transfer fails.
- A visual indicator shows if the collection has ended successfully or failed (the progress bar is green when collection is completed with no errors, or red when it fails).
- No files are written to the ISA Lite installation directory if ISA Lite is run with the -useHome option.
- Added support for Windows 7 and Linux RedHat 64-bit.
- Added inventory collection support for Solaris.

Continue to monitor the various support-related Web sites, as well as this column, for news about other tools as we encounter them.

And now, on to our main topic...

New year, new approach

Since The Support Authority began, this column has presented several of the major initiatives and tools that have been (and continue to be) developed within the IBM WebSphere Support community. In the new year, we'll be taking a slightly different approach. In addition to articles on utilities and resources that can help enhance your support and self-help experience, we will also present several articles rooted in our experience in our customer facing roles. We begin here with one of the most common "worst practices" that we encounter: the insufficient test environment.

Infrastructure malpractice

The IT industry places a good deal of focus on establishing design patterns and practices that help manage the complexity associated with enterprise scale projects. IBM's clients benefit by following such industry best practices. The emphasis on the positive, however, means that users are often not exposed to the details of what makes IT infrastructures unstable until they experience a business level outage

firsthand.

"Web site down" can be a nightmare for any online business. Besides any potential revenue loss, an outage can affect a vendor’s integrity and possibly open the door for competitors. Many technology companies rank stability as the most important factor when they choose products, even though many do not always have a clear understanding of the actual cost of product outage.

Worst practices, or in some case "malpractices," if you will, are exactly those factors that make an IT infrastructure unstable and lead to business level outages. No two business level outages are ever identical, but they do often share common factors between them. Here is a list of some of the most common and costly of those factors:

- Test environment differs from production.
- Communication breakdown.
- Blind to application state.
- Changes put directly into production.
- Insufficient capacity or scalability plan.
- Incomplete or insufficient product knowledge.
- Incomplete or outdated architecture plan.
- Vague production traffic profile.
- Inadequate load or stress testing.
- Incomplete record of changes.
- No migration plan.

Organizations commit such malpractices essentially for the same reasons: the presumption that performing the best practice is too costly or not necessary, and that cutting corners will save time and money. In reality, an organization might somehow realize a short-term savings, but there are inherent risks that can inevitably lead to business level outages and long-term losses that can far exceed any short-term savings.

Most IBM WebSphere Application Server users have some type of test environments. However, many of these environments are not technically identical to the production system because of the cost factor. As perhaps you can imagine, we have seen cases where the lack of a separate test environment has caused a product outage. During our visits to client sites over the past several years, we have recorded occurrences of malpractices, such as those listed above, if they directly caused or exacerbated a business level outage. Our analysis has showed that the

most frequent factor contributing to a highly critical situation experienced by a WebSphere Application Server user was that either the test system was not technically identical to the production system, or that a test system did not exist at all. This was the case in over 35% of all the situations we visited –- and is trending up.

The remainder of this article looks at this particular malpractice, and why this situation can become a big deal.

When your test environment differs from production

This situation includes:

- The test environment is too small, or it is overcommitted and not available when it is needed.
- The test hardware, network, or software levels differ from the production environment.
- The z/OS® LPAR on the same machine or network is not isolated from the production system.
- The configuration settings for the test systems are different from settings for the production systems.

Let's look at a sample case study to better understand why this is a malpractice.

A major bank discovered that their WebSphere Application Server for z/OS environment experienced frequent product outages during tax season, which disrupts online tax business for their customers, which in turn is harmful to the bank's relationship with its customers. The IBM WebSphere SWAT Team investigated and found many factors that contributed to this situation, primarily configuration errors.

One mistake in particular was that the bank had created two application servers on a single installation of WebSphere Application Server base: one of the application servers was the test server and the other was the production server. Because both application servers ran on the same base WebSphere Application Server, their logs are shared, their ports are configured to avoid conflict, and, most importantly, because they ran on the same binary codebase, any upgrade to the Software Development Kit (SDK) would disrupt both application servers. While frequent updating to the test system is necessary, the repeated disruption to the production system was intolerable.

Recognizing this problem, the SWAT Team urged the bank to separate the application server test and production systems on different logically partitioned

modes (LPAR) so that they would not affect each other. Any subsequent problem determination necessary after the separation is completed would be exceptionally easier.

Some developers fail (or choose not) to create a separated test system for seemingly practical or economic reasons. It might be convenient to have the test and production system together, or it might look wasteful to assign a dedicated test system. You must understand that a convenience such as this comes at a price -- and a production system outage can be very costly.

Possible solution and preventive care

Simply, you should have a separate test system that is identical to your production system. You can conduct load and stress tests on the separate test system before moving the application to production, where it should expect to work without disruption. Having a separate test system has many advantages, including:

- Prevents unintended production disruption from test activities.
- Provides a platform for performing functionality and integrity tests before performing major upgrades.
- Provides an environment for duplicating production problems and testing fixes.

Approaches for creating a test environment vary in cost, planning, and risk. Having the ability to test the application environment -- not just the application -- is a necessity. To truly benefit from a separate test environment, it is imperative that you:

- Understand that a test system is a must, not a luxury.
- Do not use a production system for test purposes.
- Physically separate a test system from a product system and, ideally, have two different groups of operators manage these systems.
- Clearly label the test and production systems with a different set of security identities.
- Install identical software on the test and production systems. Similar hardware for the both systems is preferable.
- Upgrade WebSphere Application Server and the applications themselves on the test system first before upgrading the production system.

From a practical standpoint, the type of test environment you have will depend on the the availability of resources. Starting with the most desirable, variations include:

- Maintaining an exact duplicate of the production environment.
- Maintaining a scaled-down environment with load generators that duplicate the expected load.
- Maintaining a model or simulation of the environment.

In all cases, you should implement and test all changes in your test environment before you consider adding them to your production environment.

Conclusion

Extensive cost cutting can sometimes cause businesses a loss rather than a savings. This article explained how having a separate test environment that is equivalent to the production environment is fundamental to any successful information technology project. In a rapidly moving world, it is difficult enough to follow a meticulous process and keep a record of changes, both large and small, but maintaining separate purpose-built environments not only makes it easier to keep track of such changes, it greatly reduces the risk of implementing and using them in production.

Resources

Learn

- Common Malpractices – Eleven risks to system stability and reliability
- The Support Authority: If you need help with WebSphere products, there are many ways to get it
- IBM Software product Information Centers
- IBM Software Support Web site
- IBM Education Assistant
- IBM developerWorks
- IBM Redbooks
- WebSphere Software Accelerated Value Program

Get products and technologies

- IBM Software Support Toolbar
- IBM Support Assistant

Discuss

- Forums and newsgroups
- Java technology Forums
- WebSphere Support Technical Exchange on Facebook
- Global WebSphere Community on WebSphere.org
- Follow IBM Support on Twitter!
    - WebSphere Electronic Support
    - WebSphere Application Server information
    - WebSphere Process Server
    - WebSphere MQ
    - WebSphere Business Process Management
    - WebSphere Business Modeler
    - WebSphere Adapters
    - WebSphere DataPower Appliances
    - WebSphere Commerce

- IBM Support Assistant Tools

About the author

Dr. Mahesh Rathi

Dr. Mahesh Rathi has been involved with WebSphere Application Server product since its inception. He led the security development team before joining the L2 Support team, and joined the SWAT team in 2005. He thoroughly enjoys working with demanding customers, on hot issues, and thrives in pressure situations. He received his PhD in Computer Sciences from Purdue University and taught Software Engineering at Wichita State University before joining IBM.

Innovations within reach: Enhance your ESB with the REST Gateway feature in the WebSphere DataPower XC10 Appliance

Skill Level: Intermediate

Charles Le Vay (ccl@us.ibm.com)
Senior Software Architect
IBM

Thomas Gissel (gissel@us.ibm.com)
Team Lead, WebSphere Execution Team
IBM

Lan Vuong (lvuong@us.ibm.com)
Technical Evangelist
IBM

02 Feb 2011

The IBM® WebSphere® DataPower® XC10 Appliance now expands the range of clients able to access simple data grids with the release of the REST Gateway feature. Non-Java™ based clients with HTTP capabilites, including PHP and .NET clients, can utilize the XC10 appliance for elastic caching via the REST Gateway. This article provides an overview of the REST APIs and discusses how an XC10 appliance can be integrated with a WebSphere DataPower XI50 Integration Appliance as a side cache to reduce response to response time to the clients, and improve total system throughput.

*Each installment of **Innovations within reach** features new information and discussions on topics related to emerging technologies, from both developer and practitioner standpoints, plus behind-the-scenes looks at leading edge IBM® WebSphere® products.*

Introduction

The 1.0.0.4 release of the IBM WebSphere DataPower XC10 Appliance firmware introduces a new REST Gateway feature. The REST Gateway provides non-Java based clients access to simple data grids using a set of HTTP-based operations. This new feature expands the range of clients capable of utilizing the XC10 appliance for elastic caching to any client with HTTP capabilities, including PHP and .NET clients. Using the REST Gateway feature, the XC10 can be used as a service-oriented architecture (SOA) results side cache for the IBM WebSphere DataPower XI50 Integration Appliance.

Using the XC10's simple data grid as a side cache for an XI50 can significantly reduce the load on the back end systems by eliminating redundant requests to the back ends, improving the response time to the clients and increasing total system throughput. This article will highlight the REST Gateway feature of the XC10 and provide a high-level overview of XI50/XC10 integration.

The XC10 REST Gateway feature

The IBM WebSphere DataPower XC10 Appliance is designed to be the drop-in elastic cache for your enterprise infrastructure. The XC10 appliance is the combination of the robust WebSphere DataPower hardware platform and state of the art IBM distributed caching technology.

The XC10 supports three types of data grids:

- **Simple** data grids store simple key-value pairs.
- **Session** data grids store HTTP session management data.
- **Dynamic cache** grids store cacheable objects from applications that utilize the WebSphere Application Server dynamic cache APIs or container-based caching.

To utilize a simple data grid hosted on the XC10 appliance, Java-based applications use the IBM WebSphere eXtreme Scale ObjectMap APIs to cache serializable objects as key-value pairs. Using the REST Gateway feature, non-Java based clients, such as Microsoft® .NET clients or a WebSphere DataPower Integration Appliance XI50, can use the HTTP POST method to insert or update data in a simple data grid, the HTTP GET method to get data from the grid, and the HTTP DELETE method to delete data from the grid. The REST APIs use a very simple URI format:

```
/resources/datacaches/[gridname]/[mapName]/[key]
```

For example, if you created a simple data grid named SOAGrid on the XC10 appliance, then myXC10.ibm.com, the resulting URL to access key name

my.data.item in the default map (called SOAGrid), would look like this:

```
http://myXC10.ibm.com/resources/datacaches/SOAGrid/SOAGrid/my.data.item
```

The REST APIs use the Content-type header in the HTTP request to determine the format of the data stored in the grid. You can store multiple content types in the same grid.

The REST APIs also support the creation of **dynamic maps**. When a simple data grid is created, a default map is created with the same name as the grid. The default map does not have any time-to-live (TTL) expiration. Therefore, entries in the grid will remain in the grid until they are explicitly deleted.

Using the REST APIs, applications can dynamically create additional maps as needed. Dynamic maps can be configured with one of three map TTL templates:

- No time-to-live expiration (.NONE).
- Last update time (.LUT) expiration.
- Last access time (.LAT) expiration.

It is important to know that the first operation to a map that matches a map template -- but has not yet been created -- will result in the creation of the new dynamic map. Time-to-live values are set by adding a `ttl` parameter in the HTTP request. Below is an example of setting the TTL value to 120 seconds for key "a,key." The ttl request parameter is used when the a.key value is inserted or updated into the grid using the HTTP POST method.

```
http://myXC10.ibm.com/resources/datacaches/SOAGrid/MyMap.LUT/a.key?ttl=120
```

In order to access a data grid hosted on an XC10 appliance, a Java-based client must provide a user ID and password for authentication and to verify that the client is authorized to access the grid. This same security requirement applies to REST access to a data grid. The application client must provide a basic authentication header (which contains an authorized user's user ID and password) in the HTTP header of the HTTP request. Additionally, the REST gateway supports HTTPS protocol if transport level security is required.

When data is inserted into a map using the REST gateway, a wrapper class of type com.ibm.websphere.xsa.RestValue is used to wrap the supplied content-type and request body. Using the ObjectMap APIs, a Java client can utilize this same RestValue class to insert or get data from the map. The example in Listing 1 shows the proper use of the RestValue class to access a map and insert data from a Java client.

Listing 1

```
RestValue rv = new RestValue();
rv.setContentType(“application/xml”);
String myXml=”<customer>thomas</customer>”;
rv.setValue(myXml.getBytes(“UTF8”));
ogSession.begin();
ObjectMap map = ogSession.getMap(“MyMap.LUT”);
map.insert(“thomas”,rv);
ogSession.commit();
```

An Enterprise Service Bus (ESB) is a critical component of an SOA. The ESB connects and integrates applications, services, and business process flows at the messaging layer. It performs mediation, message transformation, routing, process choreography, and provides quality of service (security, reliable message delivery, and transaction management). The WebSphere Datapower Integration Appliance XI50 is a secure easy-to-deploy hardware ESB. The XI50 is a highly scalable integration solution in a purpose-built hardware appliance. It is capable of off-loading eXtensible Stylesheet Language Transformation (XSLT) processing, XPath routing, XML conversion, and other resource intensive tasks, such as message-level or transport-level security processing, from servers to reduce latency, improve throughput, and improve server utilization.

Using the new XC10 REST Gateway feature, you can now integrate the elastic caching tier with the ESB. Traditionally, the elastic caching tier is inserted between the application server tier and the database tier; however, in this configuration, integrated with the ESB, the elastic caching tier acts as a side cache for the ESB. A simple data grid hosted on an XC10 functions as an SOA results cache for the XI50 hardware ESB. In an SOA, all application requests pass thru the ESB before they are routed to the application. Therefore, if the result of an application request is retrieved from the elastic caching tier, the application processing and processing latency for that request are eliminated. The result is a significant decrease in response time and reduction of application processing.

To use the XC10 as a side cache, an XML proxy is defined as the first component in the XI50 processing chain. It will use a set of **caching policy** rules to determine the cacheability of each incoming request. The rules are application specific, but in general, caching policy rules can trigger on the request URI, specific XML contents within the request body, or a combination of both. The rules are defined using an XSL. The XSL is then loaded into the XI50 memory. Additionally, a set of XSLs are used to generate the appropriately formatted REST requests to the XC10 REST gateway to store or retrieve data in the grid.

For this scenario, the XC10 is configured with a simple data grid that contains two

maps. The first map, called the **request cache**, is used to cache incoming requests. The second map, called the **results cache**, is used to cache the results of each cached request. These maps can be created dynamically with the appropriate TTL template based on the application and data requirements. For security, a unique user ID and password needs to be configured on the XC10, which will be used by the XI50 to access the data grid.

Figure 1. High-level design

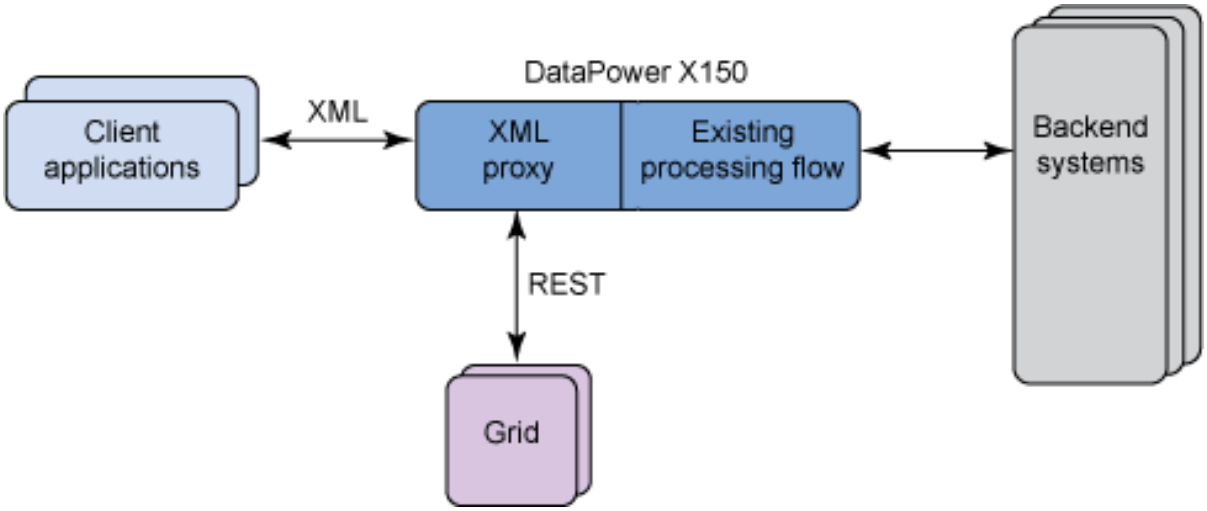


Figure 1 shows the high-level design of the XI50, using the REST APIs to utilize XC10 simple data grids as an SOA results side cache. As incoming client application requests are received, the XML proxy inspects the URI or XML body contents to determine if the request meets the criteria for being cached, based on the caching policy rules. If the request is cacheable, the XML proxy will perform a standard side cache operation. Using the REST-based HTTP GET method, the XML proxy will look to see if the request is cached in the simple data grid. If the HTTP GET returns an HTTP 404 NOT FOUND, signifying a cache miss, the XML proxy will enable the request to pass through to the existing processing flow to the application hosted in the back end systems. The XML proxy will use the REST-based HTTP POST method to insert the request into the request cache. The XML proxy will also cache the result to the result cache as it flows back through the XML proxy to the client application. If the incoming request was found in the request cache, then the result would be retrieved from the result cache, bypassing the back end systems, thus removing the latency introduced by the application and data layers.

Conclusion

Using the new REST Gateway feature, non-Java based clients can access simple data grids hosted on the IBM WebSphere DataPower XC10 Appliance. The REST Gateway expands the range of clients that can utilize elastic caching technology. As described here, integrating a WebSphere DataPower Integration Appliance XI50

with the XC10 as an SOA results side cache has the potential to significantly reduce the load on the back end systems by eliminating redundant requests to the back end systems.

Resources

Learn

- IBM WebSphere DataPower XC10 Appliance product information
- IBM WebSphere DataPower XC10 Appliance Information Center
- Video: IBM WebSphere DataPower XC10 Appliance Session Management
- Video: IBM WebSphere DataPower XC10 Appliance Monitoring
- IBM WebSphere DataPower Integration Appliance XI50 product information
- IBM developerWorks WebSphere

Discuss

- IBM WebSphere DataPower XC10 Appliance wiki

About the authors

Charles Le Vay

Charles Le Vay is a senior software architect. He recently joined the WebSphere Emerging Technologies team as a technical evangelist. His current focus is on promoting the advantages of elastic data grid technology within the enterprise. Before becoming a technical evangelist, he was the Web Service interoperability architect for IBM's WebSphere Application Server. He represented IBM on the Web Service Interoperability Organization (WS-I) Reliable Secure Profile (RSP) Working Group. As an interoperability architect, Charles focused on ensuring IBM products meet industry standard interoperability criteria. He was responsible for identifying and detailing best practices for Web services interoperability. Prior to this position, Charles specialized in mobile application development, wireless technology, and extending enterprise applications securely to mobile devices. Before joining IBM, Charles developed advanced submarine sonar systems for the Navy and specialized in signal processing and underwater acoustics. Charles is a graduate of Duke University with a degree in physics.

Thomas Gissel

Tom Gissel is an XC10 architect and its chief programmer. The XC10 is the latest in a long line innovations of which Tom has been apart. Prior to XC10 Tom was the release architect for bpmblueworks.com, WebSphere sMash's run-time lead, and the original team and technical lead for WebSphere Virtual Enterprise's Dynamic

Workload Management and Dynamic cluster components. Tom is a graduate of the University of Wisconsin with a degree in Computer Science.

Lan Vuong

Lan Vuong is currently a technical evangelist for the WebSphere elastic caching solutions. She previously worked as a software engineer for WebSphere Virtual Enterprise and was the team lead of several component areas. Lan is a graduate of the Pennsylvania State University with a degree in Computer Science.

The WebSphere Contrarian: Are you getting the most out of virtualization?

Skill Level: Intermediate

Tom Alcott
Senior Technical Staff Member
IBM

02 Feb 2011

Much has happened in the area of virtualization since the topic was first discussed in this column, so this is a good time to revisit (or re-introduce) virtualization and all its flavors to help you determine which type of virtualization could benefit your organization the most, and how to make it the most effective.

In each column, The WebSphere® Contrarian answers questions, provides guidance, and otherwise discusses fundamental topics related to the use of WebSphere products, often dispensing field-proven advice that contradicts prevailing wisdom.

What is virtualization?

It's been nearly three years since I last devoted a column to the topic of virtualization. Given the increased interest and adoption of this technology, I wanted to revisit this topic to, hopefully, provide some new perspectives, while at the same time review some aspects of this topic that haven't changed.

In its most common use today, the term virtualization refers to *"an abstraction or a masking of underlying physical resources (such as a server) from operating system images or instances running on the physical resource"* which is directly from that earlier article. Further, while recent use of this term typically refers to "server virtualization employing hypervisors," this isn't a new concept. Virtualization actually dates to the 1960s, when it was first employed on IBM® System 360-67 (I'm not young, but *no, I didn't work on those machines!*).

To provide a bit more background, a hypervisor can be classified into two types:

- **Type 1**, also known as "native" or "bare metal," where the hypervisor is the operating system or it's integral to the operating system. Examples of type 1 hypervisors would be VMware ESX and IBM PowerVM™ to name but two.
- **Type 2** refers to "hosted" or "software applications," where the hypervisor is an application running on the operating system. Some examples include VMware Server, VMware Workstation, and Microsoft® Virtual Server.

Before going on, I should point out that in this context a more precise categorization of the above is **server virtualization**. Any discussion of virtualization that covered only server virtualization would be incomplete, because you can also virtualize applications, subdividing the application servers and the resource allocation for an application (or portfolio of applications) using **application virtualization**. An example of an application virtualization technology is IBM WebSphere® Virtual Enterprise.

Is virtualization for you, and if so what type?

Chances are it's not of matter of **if** your enterprise is employing virtualization, but **how** your enterprise is employing virtualization, as well as how effective virtualization has been.

There are likely many measurements that can be applied to determine virtualization effectiveness, but because server virtualization is often viewed as a mechanism to improve server utilization, utilization would be one measure.

Another related measure is return on investment (ROI), which should increase as utilization increases. A couple of reports (see Resources) over the past few years show mixed results for server virtualization with only single digit increases in server utilization and mixed results in improving ROI. Several factors contribute to these mixed results; in some cases it's a simple lack of an effective process for tracking ROI, but another set of issues also likely contributes. This is because the partitioning of a physical server into multiple operating system images provides platform flexibility, enabling easy operating system re-provisioning or co-location as needs dictate -- and while server virtualization is excellent for server containment that provides consolidated, dedicated, and isolated environments, it also has it's drawbacks: server virtualization can be difficult to manage across an enterprise or within a data center because, typically, server virtualization is focused on individual servers.

Moreover, the real gain in increasing utilization comes from managing or distributing

the workload intelligently across many servers. It is this latter area that application virtualization can provide real benefit, because application virtualization is "application aware" in terms of application priority, application resource use, and even request and user priority -- all areas that server virtualization lacks. This awareness enables application virtualization (that is, WebSphere Virtual Enterprise) to:

- Direct application-specific requests to the server that can process them the fastest.
- Redirect application requests to another server when a given server reaches the limit of all available resources.
- Ensure that load on one application does not affect the response time of the others, in the case of co-located applications.
- Automatically detect that an error is being returned from an overloaded application and route future application requests to another server capable of running the application.
- Dynamically optimize the resource allocation (CPU and memory) for an application portfolio.

Application virtualization delivers complementary and value added benefits to infrastructures using server virtualization. Both approaches can increase utilization of servers, which in turn leads to server consolidation (and decreased costs). Application virtualization does so by intelligently managing and optimizing application workload, driving the work to the right place, and controlling the application mixture. Hardware virtualization provides its benefits by dividing a server into multiple pieces.

Optimizing virtualization delivery

Related to both server virtualization and application virtualization is a **virtualization appliance**, which according to Wikipedia is a *"minimalist virtual machine image with a pre-installed and pre-configured application (or applications) and operating system environment."*

An example technology of a virtualization appliance is the IBM WebSphere Cloudburst™ Appliance, which contains virtual machines images that are targeted at a specific virtualization container (such as VMware, PowerVM, z/VM®) and contains a pre-wired, pre-configured, production ready software stack packaged inside virtual machine images (for example, IBM WebSphere Application Server, IBM WebSphere Portal, IBM DB2®). Delivery of an image in this manner minimizes the cost and complexity of the installation, configuration, and maintenance of running complex stacks of software.

By managing not only the virtual machine image catalog, but also the placement of these images onto hypervisors, as well as the application infrastructure middleware deployment inside the images, the total virtualization environment provisioning and deployment is simplified and standardized across the organization.

Further optimizations

Since I detest the term "best practices" I'll close with some final thoughts on how best to optimize a virtualized environment. As I noted at the beginning of this article, there have been many advancements in virtualization technology in the past few years, but a number of fundamentals remain unchanged since my original article. Among these fundamentals is my advice to:

- **Never overcommit memory** -- at least not with response time sensitive Java™ applications! As explained in my earlier article, the garbage collection mechanism in Java can have a profound performance impact when memory is overcommitted. In environments that are either infrequently used or are hosting applications for which response time is not critical (for example, application development environments), some minimal amount of memory overcommit might be possible without significant impact. That said, I've worked with a number of clients over the past three years who either weren't aware of the pitfalls associated with Java application workload memory overcommit, or somehow thought that the issues related to this topic did not apply to them -- they did. While there has been some research into better integration (awareness) between hypervisors and JVMs to optimize memory allocation and swapping, commercial availability of any solution remains quite a way off.
- **Use care when partitioning CPUs.** This also still applies, but unlike memory overcommit, advances in CPU speeds and multi-core CPUs have made CPU partitioning more practical. That said, it's important to recognize that CPU partitioning doesn't create more resources, it simply enables you to divide and allocate the CPU capacity across multiple images and the application workloads running on those images. At the end of the day, there still needs to be adequate underlying physical CPU capacity to meet response time and throughput requirements when partitioning CPUs. Otherwise, poor performance will result.
- **Monitor using both operating system and hypervisor tools.** If you monitor performance only using native OS tools, such as `top` or `vmstat`, you're getting only partial and possibly misleading information. OS tools only show resource use based on what is allocated to the virtual machine, which is likely very different from the resource pools that are available on the hypervisor. If you want an accurate picture of performance on the physical server, you need to measure using the hypervisor tools, and then

use that information to adjust and optimize resource allocations to each virtual machine.

Conclusion

Hopefully you've found this re-examination of virtualization useful and will be able to employ some of the technologies described here in an optimal fashion.

Further, to show you that I'm not totally against suggesting best practices I provide one in closing: don't run with scissors. That was something my mother always told me, and we all know that our mothers are never wrong!

Resources

- The WebSphere Contrarian: Effectively leveraging virtualization with WebSphere Application Server
- WebSphere Virtual Enterprise product information
- WebSphere CloudBurst Appliance product information
- Virtualization Management Index
- Using VMware ESX Server with IBM WebSphere Application Server
- IBM developerWorks WebSphere

About the author

Tom Alcott
Tom Alcott is Senior Technical Staff Member (STSM) for IBM in the United States. He has been a member of the Worldwide WebSphere Technical Sales Support team since its inception in 1998. In this role, he spends most of his time trying to stay one page ahead of customers in the manual. Before he started working with WebSphere, he was a systems engineer for IBM's Transarc Lab supporting TXSeries. His background includes over 20 years of application design and development on both mainframe-based and distributed systems. He has written and presented extensively on a number of WebSphere run time issues.