CICS Mobile and JSON

Session 15856
Monday, 4 August 2014, 4:15 pm

Leigh Compton
IBM zGrowth Team
lcompton@us.ibm.com

Topic Abstract

- **Enterprise systems offer integrity, rollback, durability and thousands of other characteristics that make your business run without error 24/7, but enterprise systems also have a reputation of 'hiding' their data from the mobile platform. This session will demonstrate how to integrate Worklight, IBM's mobile platform for application development and application management with CICS allowing your business to expose your 'hidden' data to the mobile platform so that you can start to exploit its hidden value.**

Trademarks

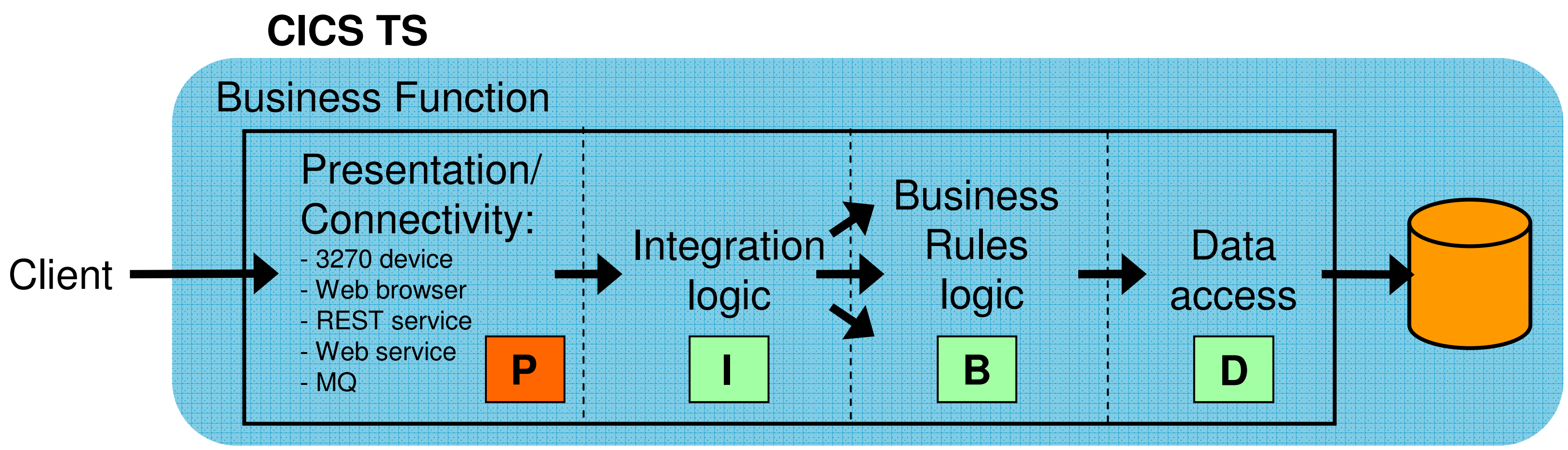
Notices

Agenda

- **Some background**
- **IBM Worklight**
- **Introducing JSON**
- **JSON support in CICS**
- **CICS and Liberty**
- **CICS TG and z/OS Connect**

Some background

CICS Application Architecture



- **Best practice in CICS application design is to separate key elements of the application, in particular:**

– Presentation logic	- 3270, HTML, XML, JSON, Servlets
– Integration or Aggregation logic	- Menu, router, tooling
– Business Rules logic	- Reusable component
– Data access logic	- VSAM, DB2, IMS, …

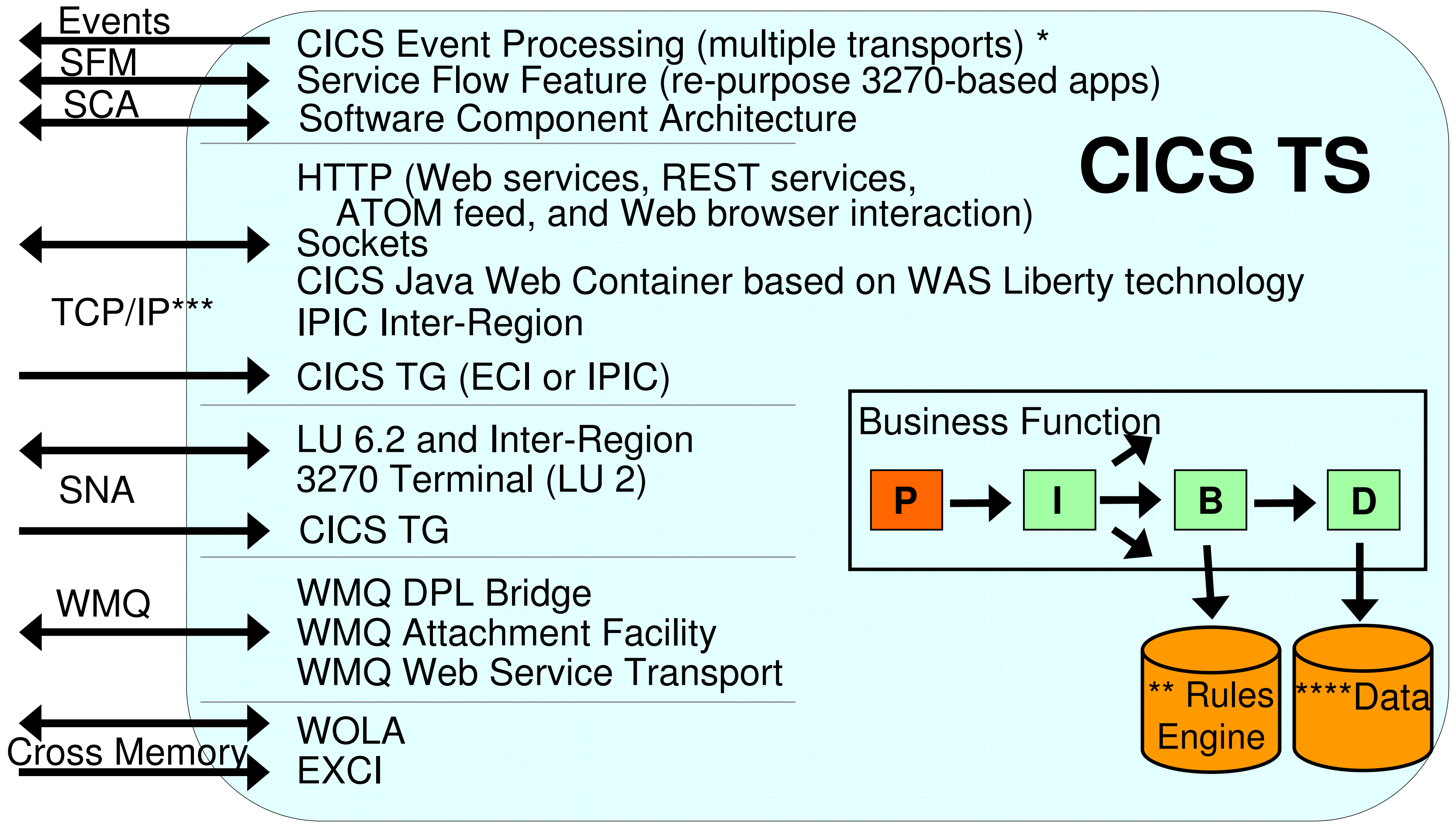
- **Provides a framework for reuse and facilitates separation of concerns, clear interfaces, ownership, and optimization**

Notes

- There is often some confusion while discussing the various Integration options with CICS as to the type of CICS program that can be invoked. For most integration options, the interface to your business logic will be via a COMMAREA or channels and containers.

  If you have very old programs where the business logic is locked behind a 3270 Interface, the Service Flow Feature (SFF) can aggregate (combine) a series of 3270 screen invocations into what looks like a COMMAREA or channel and container callable program. Additionally, you could write your own frontend to the 3270-based logic using the Link3270 Bridge or FEPI.
- New for CICS TS V5.1 is Java Servlets and JSPs. This is a CICS Java web container based on the WAS Liberty profile
- Best practices for CICS application design for several years has been to divide the presentation from the business function. Often the business function is divided into integration logic, business rules logic, and data access logic.
- If you have followed this best practice, you can easily reuse your business functions by adding logic to interface with user in a variety of ways, or by adding logic to connect to a different client or system.
- The following slides will review the various options to access date from and to CICS, and the various technologies behind the various options.

CICS Connectivity Techniques



CICS Event Processing (multiple transports) *
Service Flow Feature (re-purpose 3270-based apps)
Software Component Architecture

HTTP (Web services, REST services,
ATOM feed, and Web browser interaction)
Sockets
CICS Java Web Container based on WAS Liberty technology
IPIC Inter-Region
CICS TG (ECI or IPIC)

LU 6.2 and Inter-Region
3270 Terminal (LU 2)
CICS TG

WMQ DPL Bridge
WMQ Attachment Facility
WMQ Web Service Transport

WOLA
EXCI

* Events can have different transports or CICS can process its own events
** IBM Operational Decision Manager
*** IBM Worklight – can access CICS Data using REST, Web Services, and ATOM feeds
**** VSAM, DB2, and IMS

Notes

- WMQ, SNA, and TCP/IP are the three primary transports used to get information from/to CICS. There are various options within each transport.
- Events are emitted from CICS using one of the three primary transports, or events may be directed to CICS facilities like temporary storage. CICS can also trigger a CICS transaction. Events are listed here as being separate for ease of discussion. Events will be discussed separate from the other transports.
- The types of processing of the data over each transport is listed.
- SNA and WMQ have been supported for quite some time, so this presentation will mention them only and not provide details.
- The Cross memory options are EXCI (which has been around for a long time), and WOLA (WebSphere Optimized Local Adapter) which is cross memory communications to WebSphere Application Server (which is relatively new).

CICS TCPIP Connectivity Options

- **HTTP**
  - Web browser interfaces
  - Web services
  - REST services
  - ATOM feeds
  - Servlets and JSPs
- **Sockets**
  - Provided by Communications Server
- **IPIC**
  - Region-to-Region Communications
  - Protocol option with CICS TG
- **ECI**
  - Protocol option with CICS TG

CICS TS V4.1 added support for IPv6

Notes

- This section of the presentation will discuss connectivity options available via the TCP/IP transport
- Along with details on the connectivity option, information on the surrounding technologies will be discussed
- IPv4 is currently the pervasive technique used to assign locations on the Internet. Although it has served us well for several years, it does have some limitations. IPv4 has 32-bit addresses which allow for less than 1 billion useable global addresses. Due to the way these addresses are provisioned large companies are assign huge blocks of addresses.
- IPv6 uses 128-bit addresses which allows for 2^{128} addresses.

IBM Worklight

Mobile is a mandatory transformation

10 Billion devices by 2020

61% of CIOs put mobile as priority

45% increased productivity with mobile apps

And holds enormous opportunities

Business to Enterprise

- Increase worker productivity
- Improved claims processing
- Increase revenue with sales engagements
- Extend existing applications to mobile workers and customers
- Reducing fuel, gas, or fleet maintenance costs where relevant
- Increase employee and business partner responsiveness and decision making speed
- Resolve internal IT issues faster
- Reduce personnel cost (utilizing personal devices instead of corporate devices)

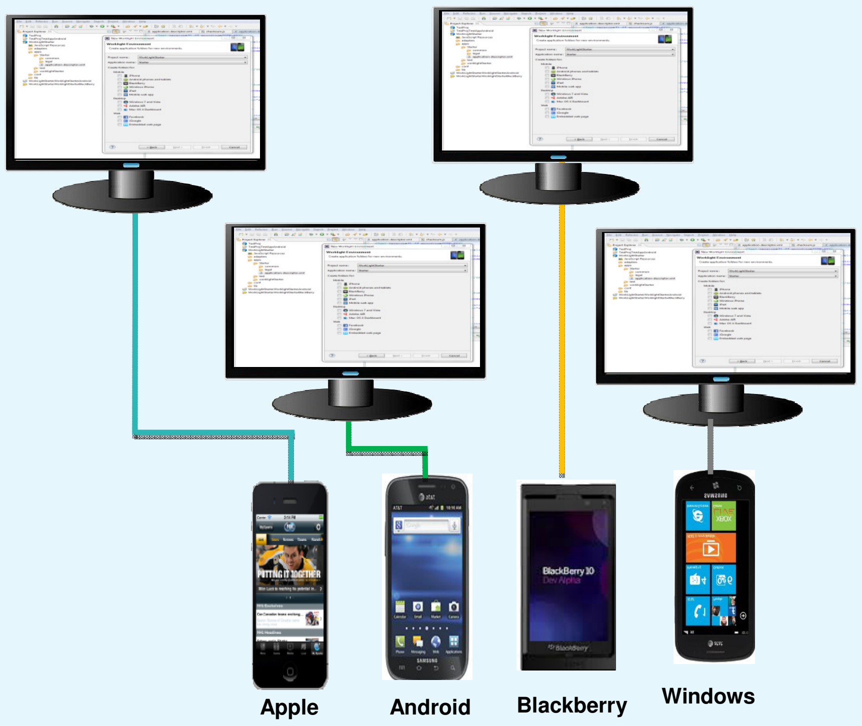
Business to Consumer

- Improve customer satisfaction
- Deeper customer engagement and loyalty
- Drive increased sales through Personalized offers
- Customer service
- Competitive differentiator
- Improve brand perception
- Deeper insight into customer buying behavior for up sell and cross sell
- Improve in store experience with mobile concierge services

Multi-platform development with a shared codebase

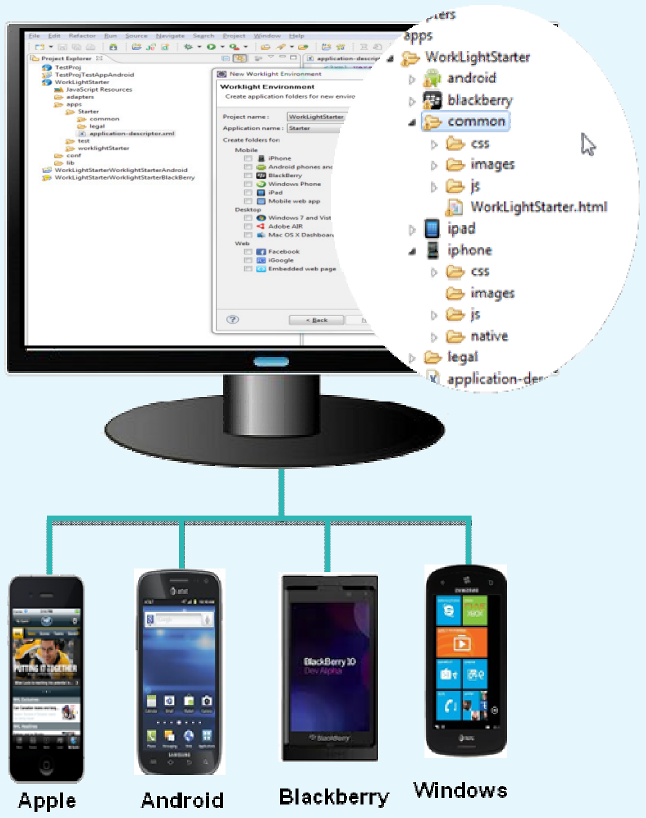
From the complexity of many…

- Multiple sets of tools & frameworks
- Four codebases to develop and maintain



To the simplicity of one

- One development environment
- One codebase to develop and maintain

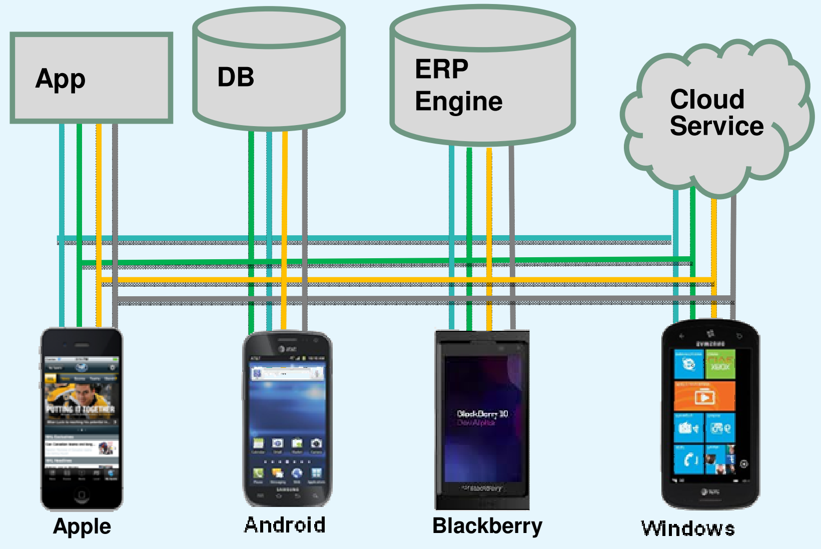


Controlled back-end integration

From multiple point-to-point integrations

•Multiple sets of integrations to enterprise resources to build and maintain

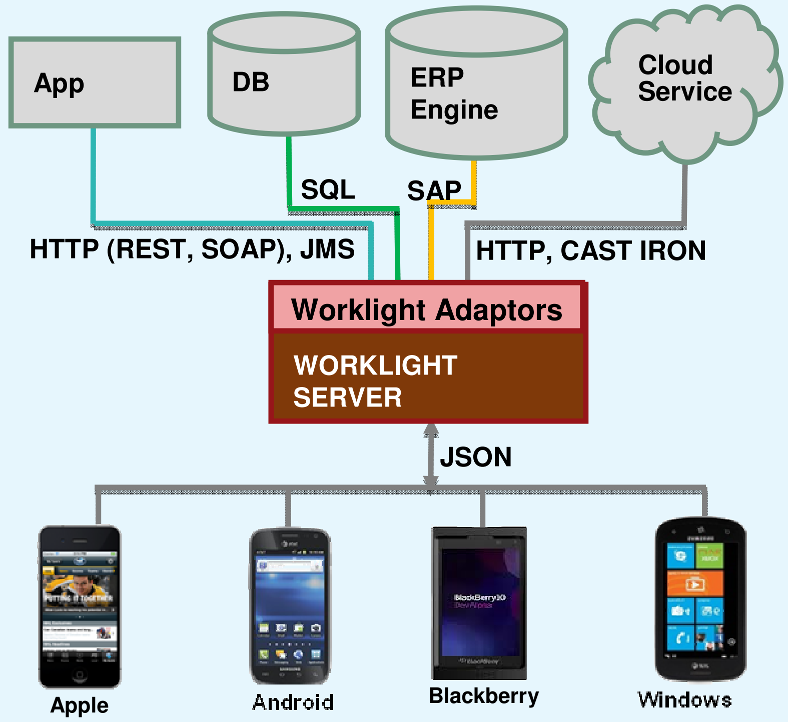
•YOU manage caching, synchronization and end-to-end encryption



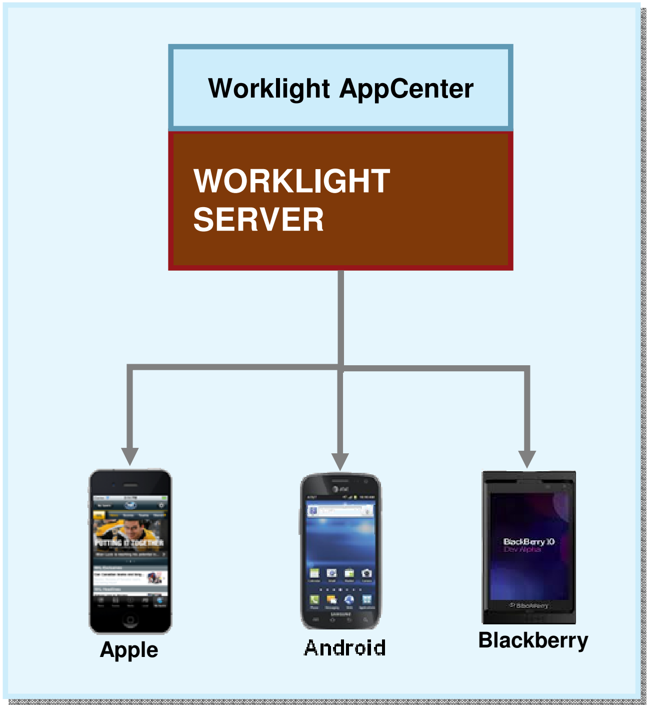
To streamlined, transparent access

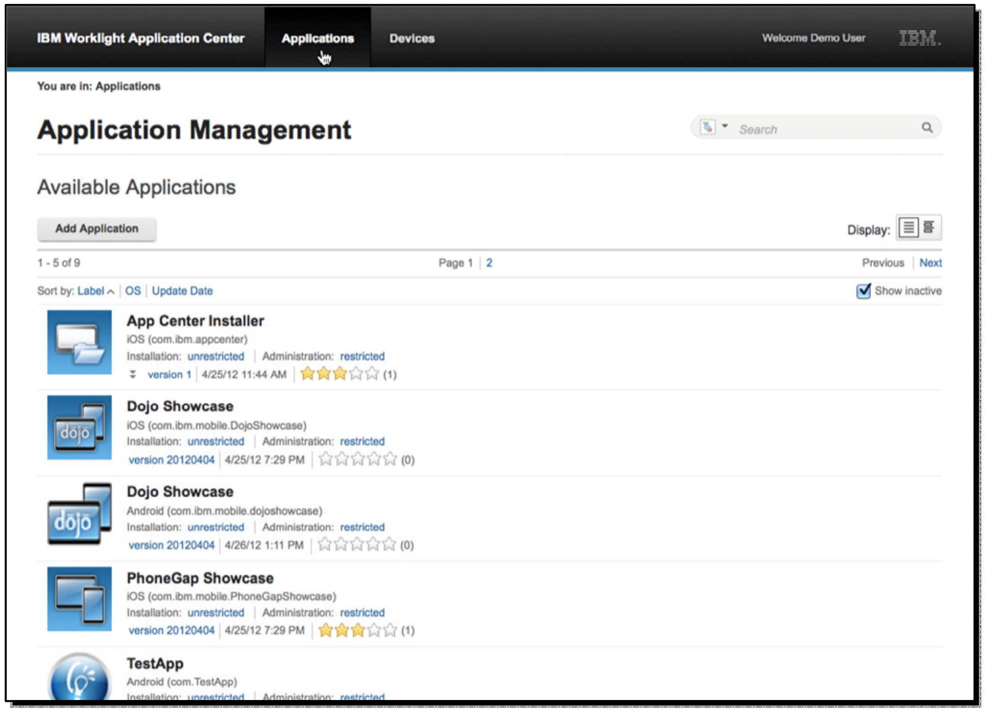
•Worklight transforms enterprise data into mobile-friendly, JSON format

•Worklight Server manages caching, data synchronization and end-to-end encryption



One App Store for all of your devices





You are in: Applications

Application Management

Available Applications

1 - 5 of 9 Page 1 | 2

Sort by: Label | OS | Update Date

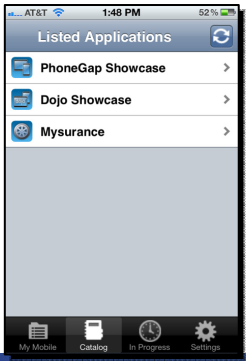
App Center Installer
iOS (com.ibm.appcenter)
Installation: unrestricted | Administration: restricted
version 1 | 4/25/12 11:44 AM (1)

Dojo Showcase
iOS (com.ibm.mobile.DojoShowcase)
Installation: unrestricted | Administration: restricted
version 20120404 | 4/25/12 7:29 PM (0)

Dojo Showcase
Android (com.ibm.mobile.dojoshowcase)
Installation: unrestricted | Administration: restricted
version 20120404 | 4/26/12 1:11 PM (0)

PhoneGap Showcase
iOS (com.ibm.mobile.PhoneGapShowcase)
Installation: unrestricted | Administration: restricted
version 20120404 | 4/25/12 7:29 PM (1)

TestApp
Android (com.TestApp)
Installation: unrestricted | Administration: restricted



Listed Applications

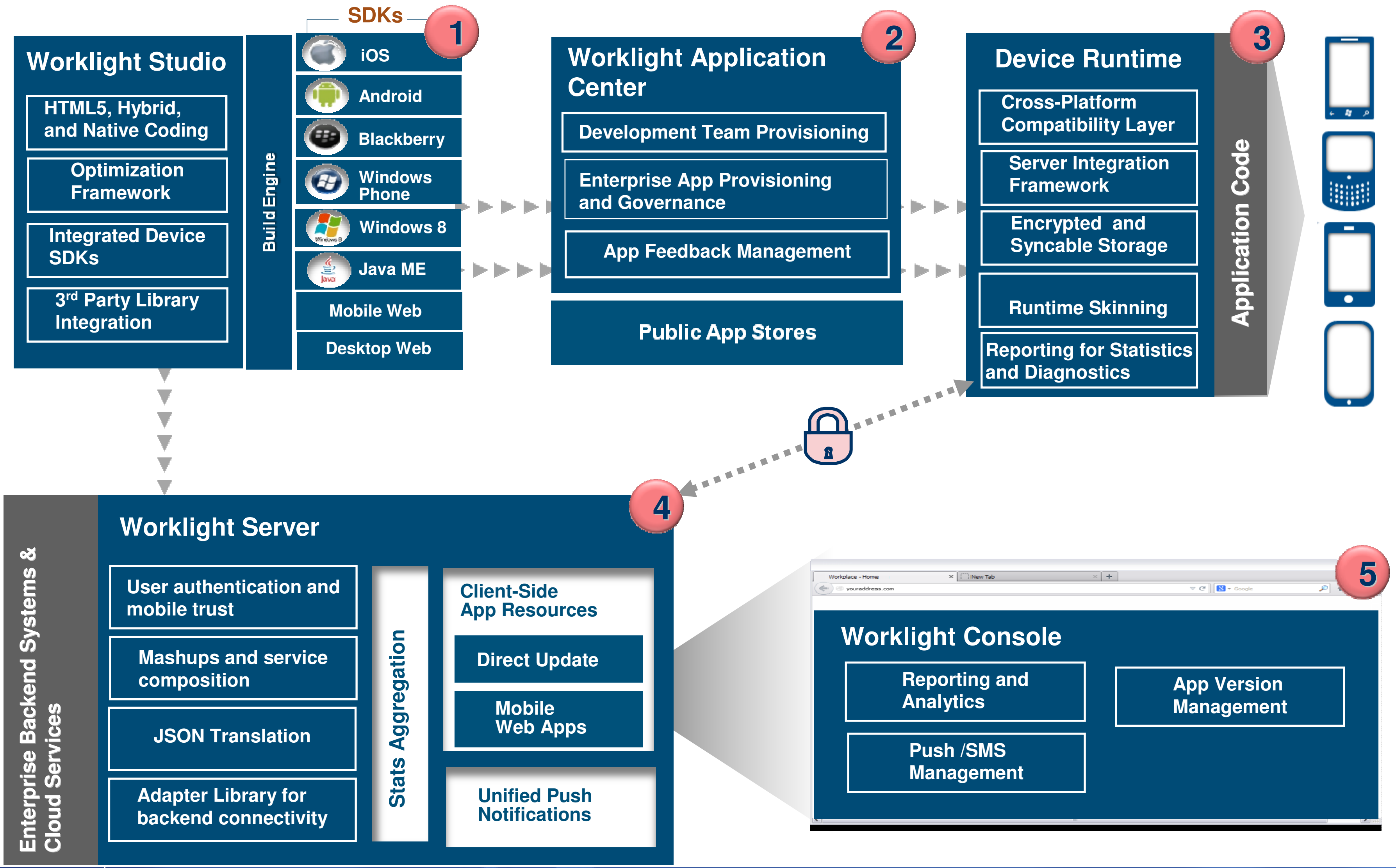
PhoneGap Showcase

Dojo Showcase

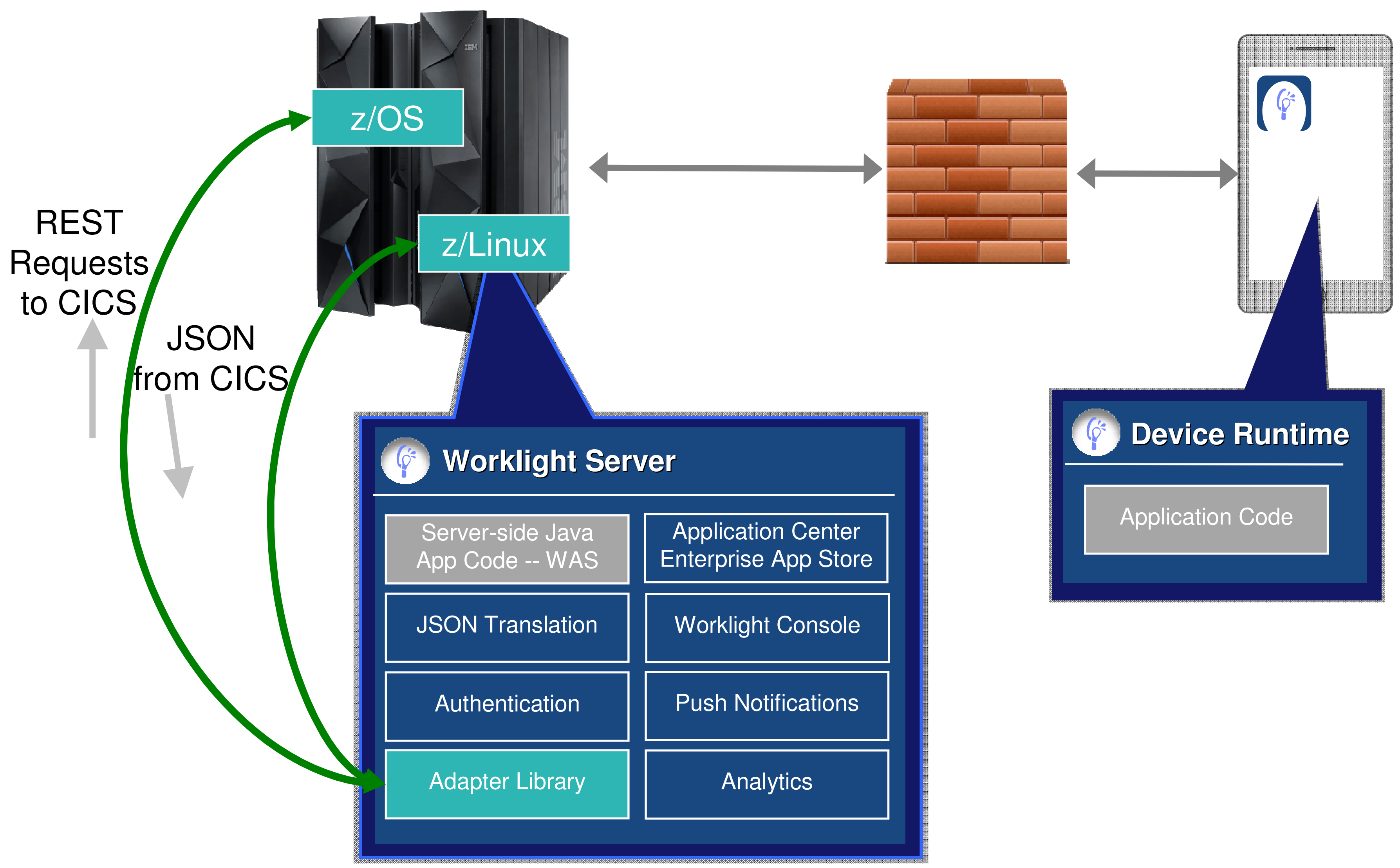
Mysurance

- **Efficient deployment of development artifacts to stakeholders for test and feed-back**
- **Easier employee enablement for up to date mobile apps access**
- **Advanced control via ACL and LDAP support**

IBM Worklight components overview


SDKs
1
Worklight Studio
HTML5, Hybrid, and Native Coding
Optimization Framework
Integrated Device SDKs
3rd Party Library Integration
Build Engine
iOS
Android
Blackberry
Windows Phone
Windows 8
Java ME
Mobile Web
Desktop Web
2
Worklight Application Center
Development Team Provisioning
Enterprise App Provisioning and Governance
App Feedback Management
Public App Stores
3
Device Runtime
Cross-Platform Compatibility Layer
Server Integration Framework
Encrypted and Syncable Storage
Runtime Skinning
Reporting for Statistics and Diagnostics
Application Code
4
Enterprise Backend Systems & Cloud Services
Worklight Server
User authentication and mobile trust
Mashups and service composition
JSON Translation
Adapter Library for backend connectivity
Stats Aggregation
Client-Side App Resources
Direct Update
Mobile Web Apps
Unified Push Notifications
5
Worklight Console
Reporting and Analytics
App Version Management
Push /SMS Management

Worklight Server on System z



Intoducing JSON

What is JSON?

- **JSON is the data structure for JavaScript**
  - Standardized as the data structure for ECMAScript (the name of the JavaScript Standard) in the late 1990s
    - Few associated capabilities – the JSON specification can be read in a few minutes
    - That’s a key part of JSON’s appeal and charm
  - JavaScript’s use has continued to increase since the mid 1990s
    - Can be used with HTML5
- **JSON support started appearing in databases in 2005 (CouchDB) and later MongoDB in 2008**
  - JSON stores were motivated by the use of JSON as an exchange format for JavaScript applications
    - Necessary to store, query, manipulate, audit the exchanged information with the UI (User Interface)
    - Increasing focus on UIs with mobile device explosion
    - JSON stores are often used in startups and by Web based companies, increasingly established corporations are looking at JSON databases
  - The “middle tier” Java layers support JSON well
  - Database architectures were/are undergoing big changes at the same time as JSON databases are being introduced, e.g., due to lower storage and memory prices
    - JSON databases have adopted new architectural approaches
    - JSON database APIs are integrated with Java and JavaScript (Not SQL : hence the term NoSQL) - programmers are the focus

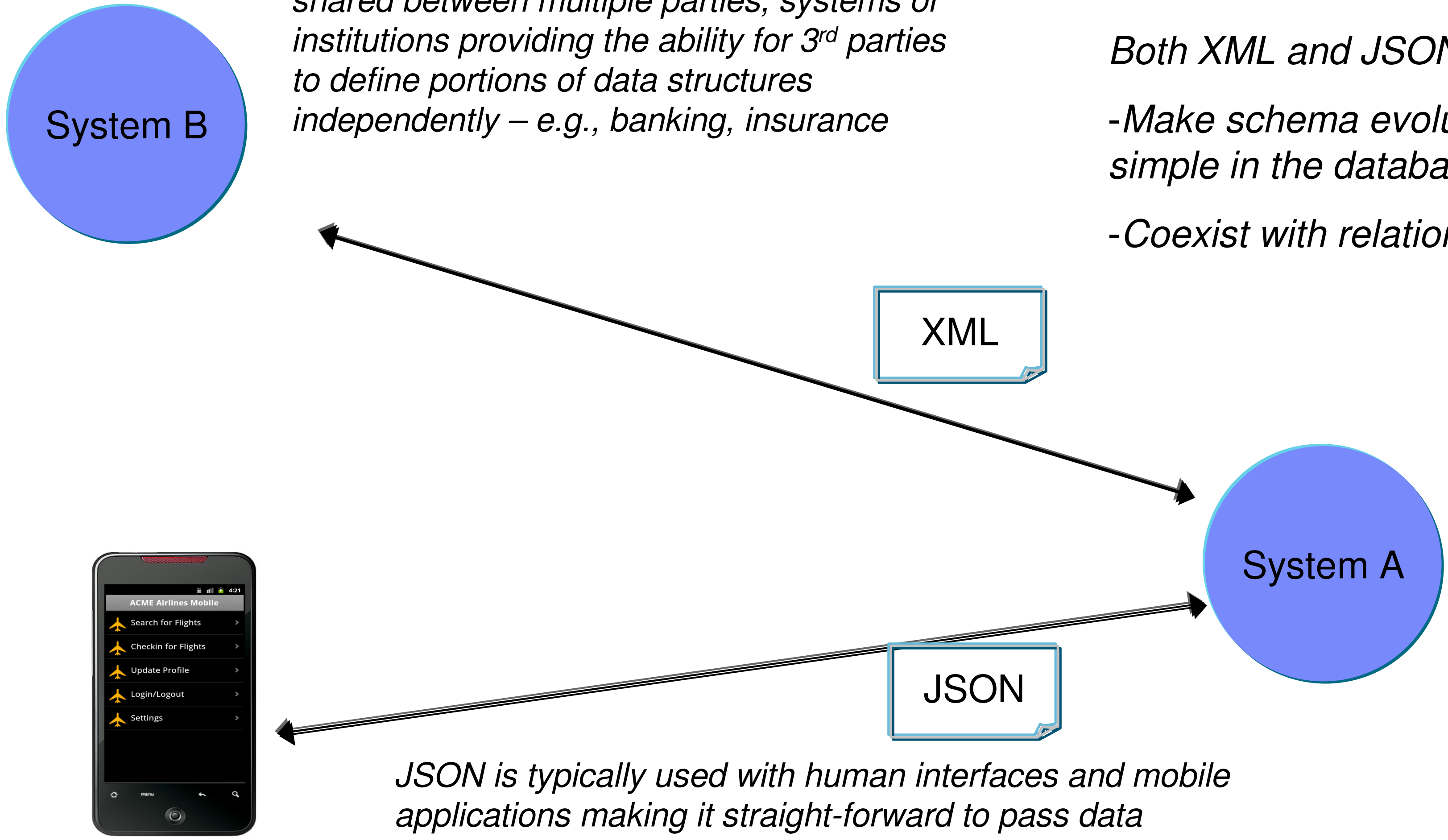
XML and JSON : Choosing between the Two

XML is typically used for data exchange or shared between multiple parties, systems or institutions providing the ability for 3rd parties to define portions of data structures independently – e.g., banking, insurance

Both XML and JSON:

-Make schema evolution simple in the database

-Coexist with relational data



JSON is typically used with human interfaces and mobile applications making it straight-forward to pass data structures back and forth

JSON Support in CICS

Notes:

- This section talks about
  - **JSON support in CICS**
  - **JSON Web services**
  - **JSON RESTful services**

CICS Support for Mobile applications

- **JSON/Mobile support**
  - New JSON assistant programs
    - Generate a JSON schema and WSBIND file from a copy book
    - Create a language structure and WSBIND file from a JSON schema
    - COBOL, PLI, C and C++
  - New linkable interface
    - Equivalent to EXEC CICS XMLTRANSFORM
    - Allows application programs to process JSON data
- **Liberty enhancements**
  - Provides support for JAX-RS and JSON class libraries

Notes:

- CICS Transaction Server for z/OS, Version 5 Release 2 provides support for web service requests with JSON and the conversion between JSON and application data. Support for JSON greatly simplifies the use of existing CICS services by mobile applications, particularly those managed by IBM Worklight® Server. You can expose CICS applications as web services with JavaScript Object Notation (JSON) payloads, create new RESTful applications, call existing JSON applications, and convert JSON from any source to and from the application data.
- This support for JSON and REpresentational State Transfer (REST) was previously available in the CICS TS Feature Pack for Mobile Extensions.
- Also in CICS TS v5.2, the Liberty web container now supports use of the JAX-RS classes

CICS Support for JSON

- **New function in CICS TS V5.2**
- **Available for CICS TS V4.2 and V5.1 via CICS Feature Pack for Mobile Extensions**
- **Allows CICS applications to be exposed as JSON based Services**
  - Request/Response mode
    - Very similar to SOAP Web Services
  - RESTful mode
    - Classic REST interaction based on HTTP method
  - Supports: COBOL, PL/I, C/C++
  - Uses a WSBind file, generated by DFHLS2JS or DFHJS2LS
  - Hosted in a CICS PIPELINE resource
- **Provides programmatic mode for JSON-copybook transforms**

Notes:

- CICS Support for JSON in CICS TS V5.2 and the CICS TS Feature Pack for Mobile Extensions build on the service-oriented architecture (SOA) capabilities in CICS to work with JavaScript Object Notation (JSON). You can expose CICS applications as web services with JSON payloads, create new RESTful applications, call existing JSON applications, and convert JSON from any source to and from application data.
- The Mobile Extensions Feature Pack version 1.0 requires CICS TS for z/OS®, Version 5.1 or CICS TS for z/OS, Version 4.2.
- JSON support is fully integrated into CICS TS V5.2
- You can obtain the following benefits:
  - You can enable CICS applications to be called as request-response style web services with a JSON payload.
  - You can host new RESTful JSON applications in CICS.
  - You can transform JSON from any source to structured application data.
  - You can call web services hosted externally using JSON.
  - You can broaden the reach of your mobile applications to include CICS data.
  - IBM® Worklight® can be used to communicate with existing CICS programs using JSON

JSON Support in CICS

- **CICS supports:**
  - JSON web service – Request/Response
    - Bottom-up – where you start with a copybook
    - Top-down – where you start with a JSON schema
      - Draft 4 of the specification at http://json-schema.org
  - JSON RESTful service
    - Top-down only
  - JSON Transformer Linkable Interface (callable)
    - Bottom-up
    - Top-down

Notes:

- CICS supports two modes of JSON web service, Request-Response and RESTful. CICS also supports a programmatic scenario in which applications can transform JSON data to and from COBOL style data formats themselves.
- The Request-Response JSON pattern is very similar to that of SOAP based web services in CICS. In this scenario the JSON client must connect to CICS using the HTTP POST method. A Request-Response mode JSON web service can be developed in either bottom-up mode or top-down mode. The Request-Response pattern may be used to build JSON Web Services that target either Commarea or Channel attached CICS PROGRAMs. A Request-Response JSON web service can be used only in provider mode (where CICS acts as the server).
- RESTful JSON web service implements the architectural principles of the REpresentational State Transfer (REST) design pattern. This design pattern is unlikely to be relevant for existing CICS applications, so is available only in top-down mode. CICS implements a pure style of RESTful application, where the data format for POST (create) GET (inquire) and PUT (replace) are the same. CICS implements a pure style of RESTful application, where the data format for POST (create) GET (inquire) and PUT (replace) are the same.
- In the programmatic mode, an application can LINK to a CICS supplied program, DFHJSON, and ask it to transform application data into JSON data, or JSON data into application data. CICS has no built-in support for requester mode JSON web services, but an application can call a remote JSON web service by exploiting the programmatic mode.

Request-Response

- **Similar to a SOAP-based web service**
  - Implemented using a PROGRAM in CICS
    - Target either Commarea or Channel attached programs
  - CICS only supports provider mode
  - HTTP client must use HTTP POST method
  - HTTP body must contain JSON data

Notes:

- The Request-Response JSON pattern is very similar to that of SOAP based web services in CICS. The web service is implemented using a PROGRAM in CICS. The PROGRAM has input and output data formats, described using language structures (such as COBOL copybooks), and CICS is responsible for transforming incoming JSON messages into application data, and linking to the application. The application returns output data back to CICS, and CICS transforms this into JSON data to return to the client.
- In this scenario the JSON client must connect to CICS using the HTTP POST method.
- A Request-Response mode JSON web service can be developed in either bottom-up mode or top-down mode. In bottom-up mode an existing CICS PROGRAM is exposed as a JSON web service using the DFHLS2JS JSON Assistant. In top-down mode a new JSON web service can be developed to implement an interface described using existing JSON schemas. In top-down mode, the DFHJS2LS JSON Assistant is used to generate new language structures, and an application must be created to use them.
- The Request-Response pattern may be used to build JSON Web Services that target either Commarea or Channel attached CICS PROGRAMs. A Request-Response JSON web service can be used only in provider mode (where CICS acts as the server).

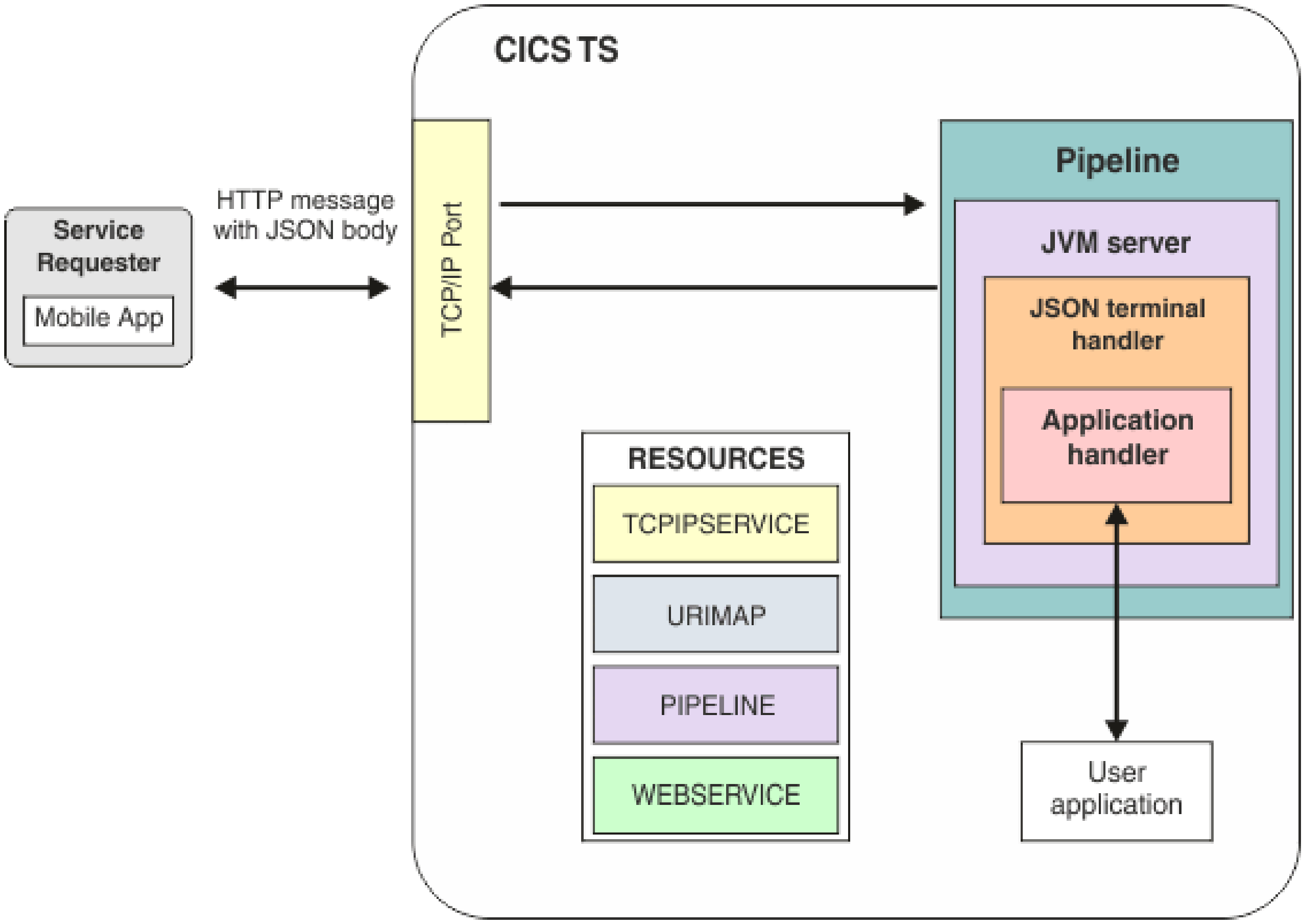
RESTful services

- **REST is a design pattern for interacting with resources**
  - Each resource has a identity, a data type, and supports a set of actions
- **The URI identifies the resource**
- **HTTP methods are used to indicate the action**
- **Classic REST actions**
  - POST: create a new resource
  - GET: retrieve a resource or a list of resources
  - PUT: update a resource
  - DELETE: destroy a resource
- **RESTful APIs may be designed with different uses of HTTP methods**
- **Traditional CICS applications are unlikely to match RESTful architecture**
  - Use of CICS JSON support for RESTful services will require new programs
  - Wrapper programs can be used to expose existing applications

Notes:

- REpresentational State Transfer, or REST, is a design pattern for interacting with resources stored in a server. Each resource has an identity, a data type, and supports a set of actions.
- The RESTful design pattern is normally used in combination with HTTP, the language of the internet. In this context the resource's identity is its URI, the data type is its Media Type, and the actions are made up of the standard HTTP methods (GET, PUT, POST, and DELETE).
- This style of service differs from Request-Response style web services:
  - Request-Response services start interaction with an Application, whereas RESTful services typically interact with data (referred to as 'resources').
  - Request-Response services involve application defined 'operations', but RESTful services avoid application specific concepts.
  - Request-Response services have different data formats for each message, but RESTful service typically share a data format across different HTTP methods.
- The four major HTTP methods define the four operations that are commonly implemented by RESTful Services. The HTTP POST method is used for creating a resource, GET is used to query it, PUT is used to change it, and DELETE is used to destroy it. The most common RESTful architecture involves a shared data model that is used across these four operations. This data model defines the input to the POST method (create), the output for the GET method (inquire) and the input to the PUT method (replace). This simple design pattern is popular within the RESTful community, but it's not the only RESTful design pattern. Some RESTful APIs are designed in other ways.
- A fifth HTTP method called 'HEAD' is sometimes supported by RESTful web services. This method is equivalent to GET, except that it returns only HTTP Headers, and no Body data. It's sometimes used to test the Existence of a resource. Not all RESTful APIs support use of the HEAD method.
- Traditional CICS® applications are unlikely to match the RESTful architectural pattern. Typical CICS applications implement multiple operations, each of which will have data models for input and output formats. These existing operations are unlikely to map directly to the four HTTP methods. For this reason the RESTful architectural pattern is primarily aimed at new applications in CICS. To expose existing CICS applications as RESTful Services you might need to wrap them with a new interface that conforms to the RESTful principles.

CICS as a service provider for JSON requests



Resources used by JSON web services

- **URIMAP is matched for the URI found in the request.**
  - URIMAP points to WEBSERVICE and PIPELINE resources.
- **WEBSERVICE resource points to WSBind file to be used for data transformation between JSON and language structures.**
- **PIPELINE resource defines any handlers that may process the message.**
  - Terminal handler is JSON handler; Java program
  - Application handler performs mapping of JSON data to/from copybook format
- **JVMSERVER resource defines an AXIS2 execution environment for processing JSON data**

Liberty Web Container in CICS

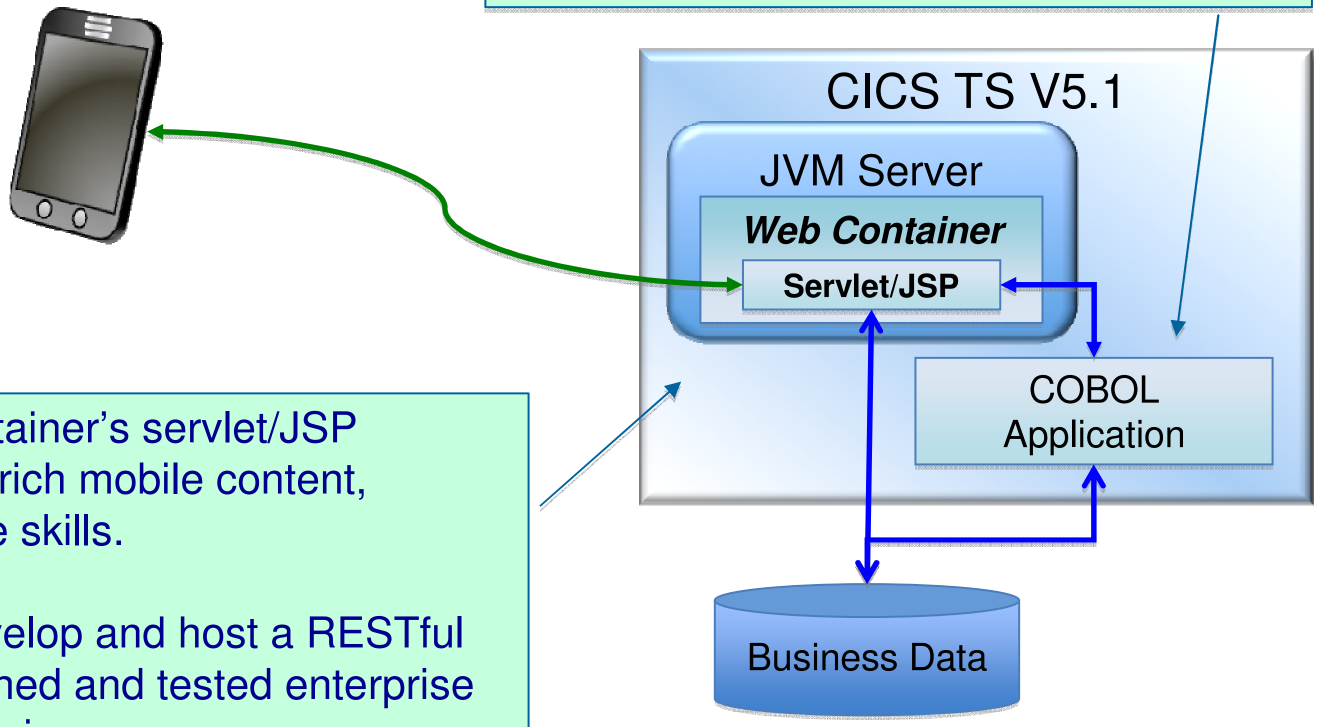
Notes:

- These slides discuss CICS's support of the Java Servlet and JSP specifications

Doing it all Java style

RESTful services can now be hosted within the CICS Web Container, with new support for the JAX-RS API

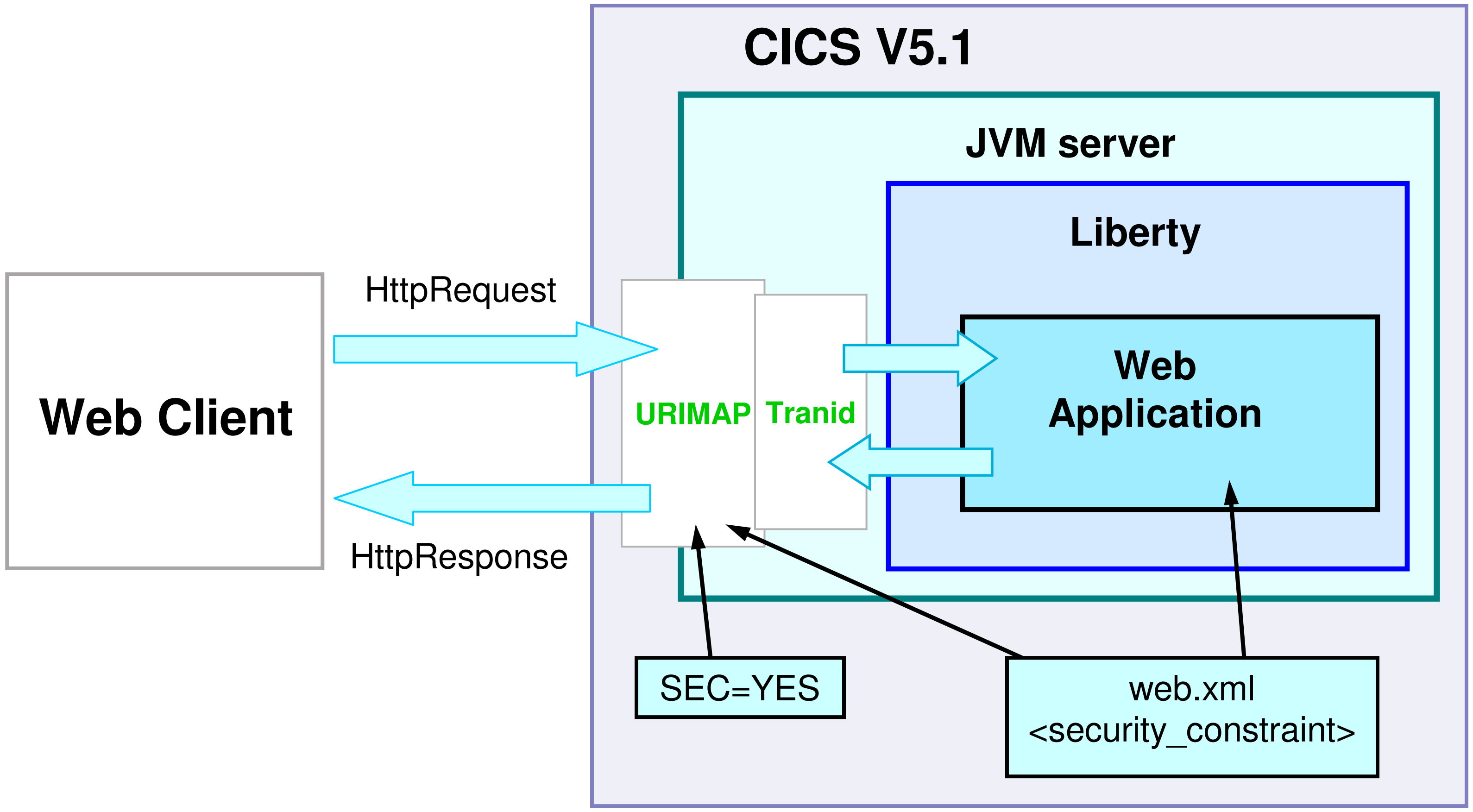
Link to existing C/C++, COBOL, PL/I, and Java applications to exploit existing enterprise applications and services



Exploit the web container's servlet/JSP features to develop rich mobile content, building on available skills.

Ideal location to develop and host a RESTful interface to established and tested enterprise applications and services

CICS Servlet and JSP support…



- **Standard Java APIs (Servlet, JSP, JDBC)**
- **Supports much of the Liberty Profile**
- **Can use the JCICS API or EXEC CICS LINK to a program in any language**

CICS Servlet and JSP support…

- For the deployment of lightweight Java servlets and JavaServer Pages (JSPs)
- CICS's support built on the WAS Liberty profile technology
- Supports servlets and RESTful clients
- Highly configurable (CICS does not support everything in the Liberty Profile)
- Can use the JCICS API and can connect to DB2
- Can use Eclipse, Rational Application Developer (RAD), and Rational Developer for System z to develop servlets and JSPs
- Start Liberty in ASCII JVMServer (JVMProfile already set up for you)
- Define BUNDLE resource for your application
- Several samples

CICS Servlet and JSP support

- Uses Java 7 (64-bit)
- If you want to secure the application with CICS security, create a web.xml in the web project to contain a CICS security constraint
- Can use CICS basic authentication
- Can run under Liberty security, but must supply your own security roles and basic user registry
- By default runs under the CJSA transaction
- Can add a URIMAP and TRANSACTION resource to a CICS bundle if you want to run under a specific transaction

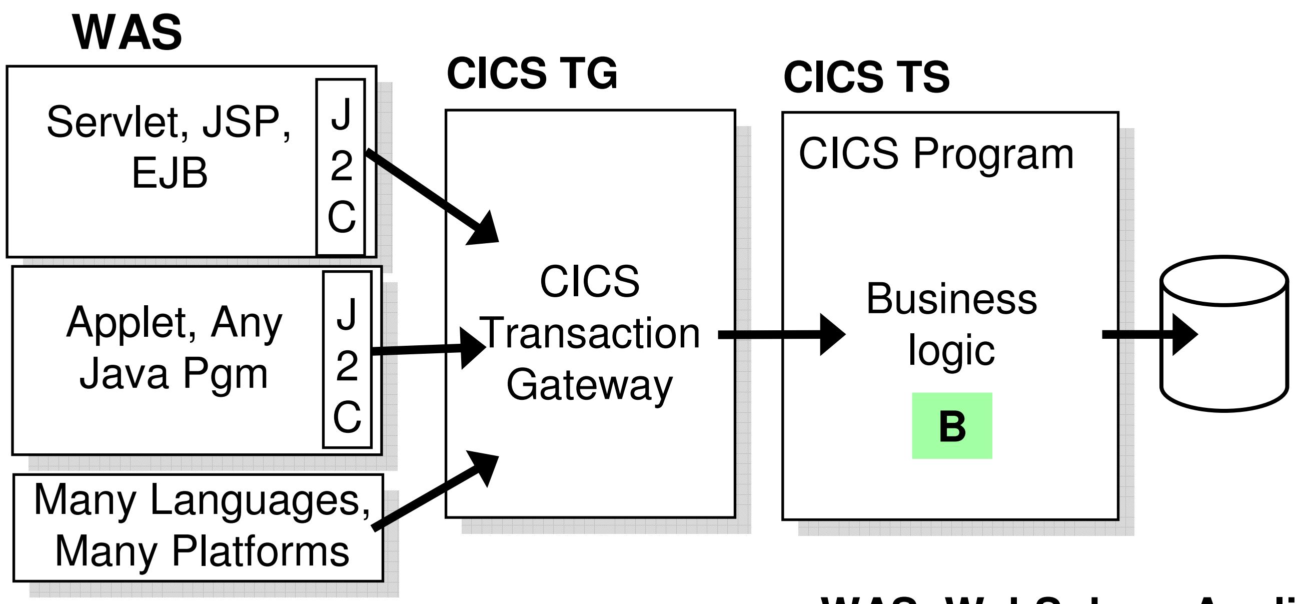
CICS Transaction Gateway and z/OS Connect

Notes:

- This section talks about
  - CICS Transaction Gateway
  - z/OS Connect

CICS Transaction Gateway

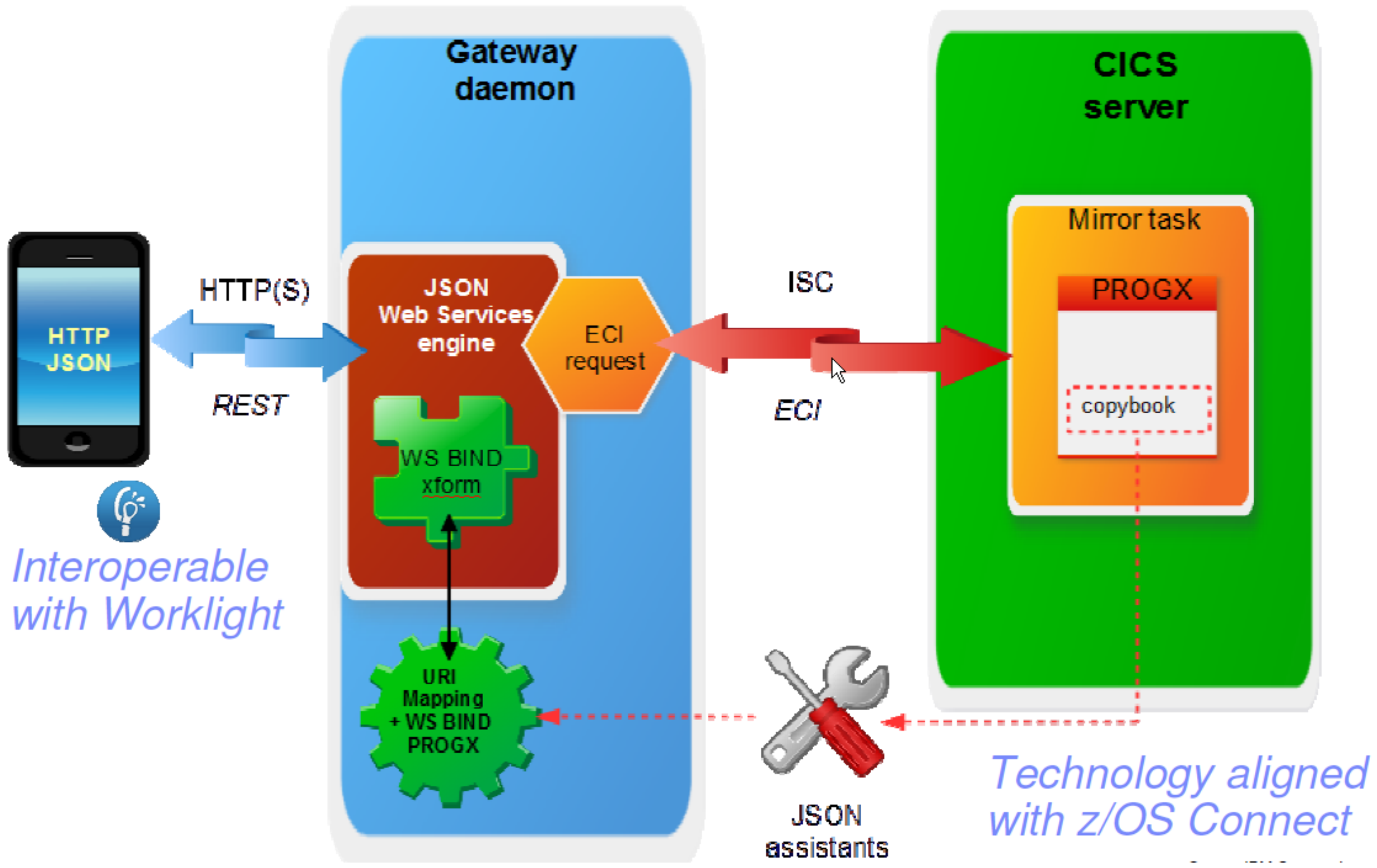
- Enabler for the Java Connector Architecture (J2C)
- Using C, C++, C#, VB, COBOL on workstation
- COMMAREA or channel and containers
- Supported with any JEE 1.4 or higher compliant application server
- SNA or TCP/IP (ECI, IPIC) to CICS



WAS=WebSphere Application Server

JSON services in CICS TG

- **Delivered in CICS TG V9.1**
  - Open beta available now



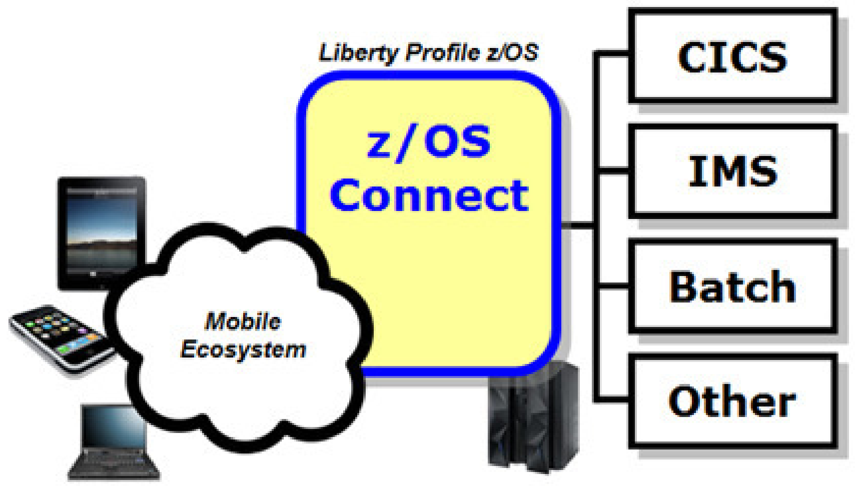
CICS TG JSON services - overview

- **New style of remote client and data representation**
  - No client-side IBM code required
  - Active data transformation within the Gateway daemon
  - Service-enablement for any release of CICS server
  - Exploit the high availability and instrumentation features of CICS TG
- **Top-down style service enablement**
  - Generate COBOL, C, PL/1 language structures from a JSON schema
  - Non-RESTful can be used with COMMAREA of channel programs
  - RESTful must use channel programs (and therefore IPIC)
- **Bottom-up style service enablement**
  - Generate JSON schema from COBOL, C, PL/1 language structures
  - Target program is not REST-aware, so JSON web service is non-RESTful; COMMAREA or channel programs supported

CICS TG JSON services - overview

- **Powered by Liberty, compatible with z/OS Connect and CICS TS JSON support**
- **Based upon proven technologies**
  - Uses a “private” embedded WebSphere Liberty profile within the Gateway daemon for the HTTP server
  - Uses common data transformation components at run-time from CICS TS for z/OS
  - JSON ws-bind files are interoperable with CICS TS, CICS TG for z/OS V9.1 open beta and z/OS Connect solutions
- **The JSON web services assistant is included with CICS TG for z/OS V9.1 open beta**
  - Uses common tooling components with a simplified interface for CICS TG

Why z/OS Connect?



- z/OS Connect simplifies and makes the environment more consistent and manageable

- Provides a common and consistent entry point for mobile access to one or many backend systems
- Java, so runs on specialty engines
- Shields backend systems from requiring awareness of RESTful URIs and JSON data formatting
- Provides point for authorization of user to invoke backend service
- Provides point for capturing usage information using SMF
- Simplifies front-end functions by allowing them to pass RESTful andJSON rather than be aware of or involved in data transformation

z/OS Connect delivery

- **WAS z/OS**
  - Delivered as function that runs inside Liberty Profile z/OS. Initially will use WOLA (WebSphere Optimized Local Adapters) to access backend.
- **CICS** *
  - Delivered as part of Liberty Profile that runs inside of CICS region. Will use JCICS LINK interface to access CICS programs.
- **CICS Transaction Gateway** *
  - Delivered as part of Liberty Profile to run inside the CICS TG. Will use ECI/DPL interface to access CICS programs.
- **IMS** *
  - Initially this ends up looking just like the WAS z/OS approach: that is, Liberty Profile z/OS with z/OS Connect inside. Difference is this z/OS Connect will be able to talk to IMS Connect for access into IMS

* **Statement of direction, not yet available**

Data transformation

- **Bind files**
  - Identical to JSON binding files used by JSON feature of CICS
  - Generated with a supplied utility
    - z/OS Connect ships utility programs
    - Or use CICS-provided utilities
  - Provide z/OS Connect with knowledge of how JSON maps to the target data structure

Summary

- **CICS offers multiple ways to integrate CICS applications with mobile devices**
  - Newest support is for JSON data formats
    - Automatically and programmatically using JSON support
      - In CICS TS
      - In CICS TG
      - In z/OS Connect
    - Programmatically using Java Servlets, JSP, or JAX-RS
    - Programmatically using CICS Web Support API

REST Architecture

Notes:

- This section of the presentation discusses REST (REpresentational State Transfer).

REST Services

- **Defined by Roy Fielding in 1994, documented in his year 2000 doctoral thesis**
- **Similar in concept to hyperlinked data**
- **Lightweight data transfer**
- **Representational State Transfer**
  - Nouns (URLs) indicate what is being worked on
  - Verbs (GET, PUT, POST, DELETE) indicate the action to be performed (List, Create, Read, Update, Delete)
- **Format of results is not defined**
  - Popular formats of returned data are XML and JSON
- **Approaches in CICS**
  - JSON Service feature of CICS TS V5.2
    - Available as CICS Feature Pack for Mobile Extensions for CICS TS V4.2 and V5.1
  - Servlet and JSP engine (subset of Liberty) in CICS TS V5.1 and V5.2
  - Can use EXEC CICS TRANSFORM for XML parse/create
  - CICS Web Support and WEB API
  - ATOM Feed (CICS TS V4.1+)
  - Dynamic Scripting (CICS TS V4.1 and V4.2, CICS TS V5.1 and V5.2)

Notes:

- REST (REpresentational State Transfer) is an architectural style that applies the approach we use to access Web pages to access our business data. Just like we use a URL to access the current state of a Web page, you use a URL to access the current state of business data. We can specify a specific Web page on a URL, we can also specify a specific account number on a URL.
- We normally need to perform LCRUD (List, Create, Read, Update, and Delete) functions on our business data. The HTTP ‘methods’ that flow with the request indicate the action to be performed on the data. Whereas we normally only use a GET or a POST method when accessing a Web page, for data, a GET method indicates a list or a read, DELETE for a delete, POST for an add, and a PUT for an update.
- REST results in very lightweight interactions with a minimal amount of characters transferred.
- The format of the returned data is not dictated, although most people use XML or JSON (JavaScript Object Notation.
- REST is documented in Roy Fielding’s year 2000 doctoral thesis. In his thesis, Fielding indicates that REST started in 1994 and was iteratively redefined. Since many people were not aware of REST, they think it is a follow-on to Web services, however Web services came after REST.
- For situations where you want interfaces documented with WSDL, transactionality, and more security options, Web services are great. Where you just need lightweight data access, REST is great.
- One of the primary uses of REST is for requests from Web browsers. JavaScript running in a Web browser can use AJAX (Asynchronous JavaScript and XML) to make RESTful requests to backend data and business logic systems such as CICS.
- The easiest ways to expose business data or business logic in CICS as RESTful services is use of the CICS WEB API, ATOM feeds. These will be discussed in upcoming slides.

The URL - Directing Requests using HTTP

- **Scheme – http or https**
- **Host name or address**
- **Port (optional)**
- **Path**
- **queryString (may or may not see these on URL)**
  – Name=value pairs separated by an ‘&’

```
http://demomvs.demopkg.ibm.com:8091/account?name=Dennis&state=TX

scheme://host:port/path?queryString
```

Notes:

- HTTP requests are routed using a URL. A URL (Uniform Resource Locator) specifies the location of the server, and optionally, some information that should be passed to the server.
- The URL consists of a scheme, which can be http or https. An ‘http’ specification indicates that the flows are not to be encrypted, but an ‘https’ specification indicates the flows should be encrypted.
- Next comes the host which is a DNS (Domain Naming Service) name, or a dotted-decimal address. A DNS name is just a ‘friendly’ name for a dotted-decimal address. When a DNS name is specified, a DNS server will be contacted to resolve the specified name to dotted-decimal address.
- The (optional) port is separated from the host name by a colon. If no port is specified and the scheme is http, the port used will be 80. If no port is specified and the scheme is https, the port used will be 443.
- Following the optional port is the path. The path can be from 1 to 255 characters, and its function is to tell the server what to do with the request.
- Following the path is an optional query string. If a query string is present, it is separated from the path by a ‘?’ character. The query string is a set of name/value pairs (separated by an ‘&’) that are to be sent to the server.

REST Service Requests

Request URI	HTTP Method	Event
Collection URI, e.g.: `http://xyz.com/prefix/myResource`	GET	List
	POST	Create
	PUT	putCollection
	DELETE	deleteCollection
Member URI, e.g.: `http://xyz.com/prefix/myResource/resourceID`	GET	Retrieve
	POST	postMember
	PUT	update
	DELETE	delete

Notes:

- This illustrates the relationship between the URL, the HTTP method, and the action to be taken on the data.
- For example, if the URL was http://www.books.are.us/JKRowling with a GET method, you would be asking for a list of books written by J. K. Rowling.
- If the URL was http://www.books.are.us/JKRowling/HarryPotterAndThePhilosophersStone with a GET method, you would be asking for details on the first book in the Harry Potter series.
- As you can see, REST style requests are lightweight.

REST Simple Sample

- **Request**

```
GET /mortgage/231677 HTTP/1.1
  Host: www.example.com
  Accept-Language: en
  Charset: UTF-8
```

- **Response**

```
HTTP/1.1 200 OK
  Language: en_us
  Charset: UTF-8
  Content-Type: text/json
{“principal”:”238000”,”rate”:”3.5”, “type”:”5/1 ARM”}
```

or

```
HTTP/1.1 200 OK
  Language: en_us
  Charset: UTF-8
  Content-Type: text/xml
<mortgage><principal>238000</principal><rate>3.5</rate><type>5/1 ARM</type></mortgage>
```

Notes:

- This page illustrates the flow of data ‘on the wire’ for a REST request.
- The samples include the HTTP (HyperText Transfer Protocol)
- The request starts with an HTTP request line (“GET /mortgage/231677 HTTP/1.1”) followed by HTTP headers that define additional characteristics of the transmission.
- The request indicates that we would like the details on mortgage 321677
- Since the most popular way of returning information is via XML or JSON, we have shown an example of each.
- In both situations there is an HTTP response line indicating the status of the request plus HTTP headers indicating the type, language, and codepage of the request (you may see additional header information). In the body of the HTTP transmission is the application data either in XML or JSON notation.
- Some people feel that JSON has less characters to transfer and is easier to parse so it is a popular format.
- REST services can be implemented in CICS using the CICS WEB API, using ATOM feeds, and using Servlets/JSPs.