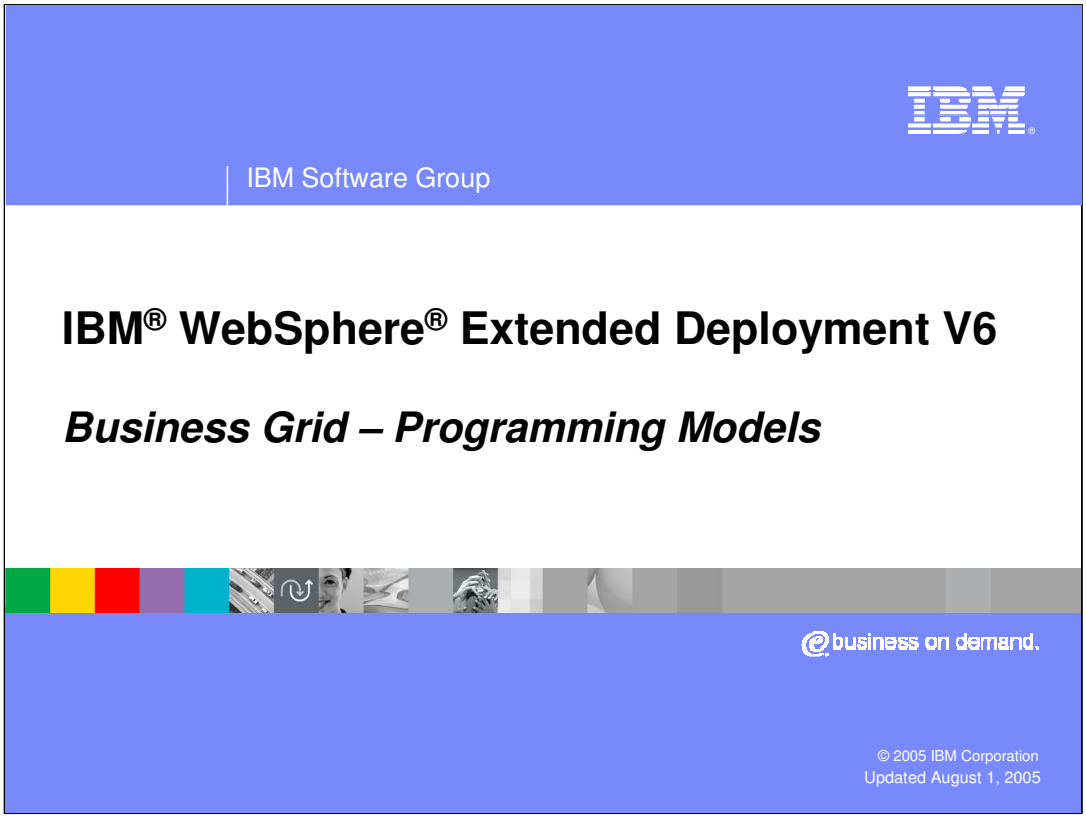
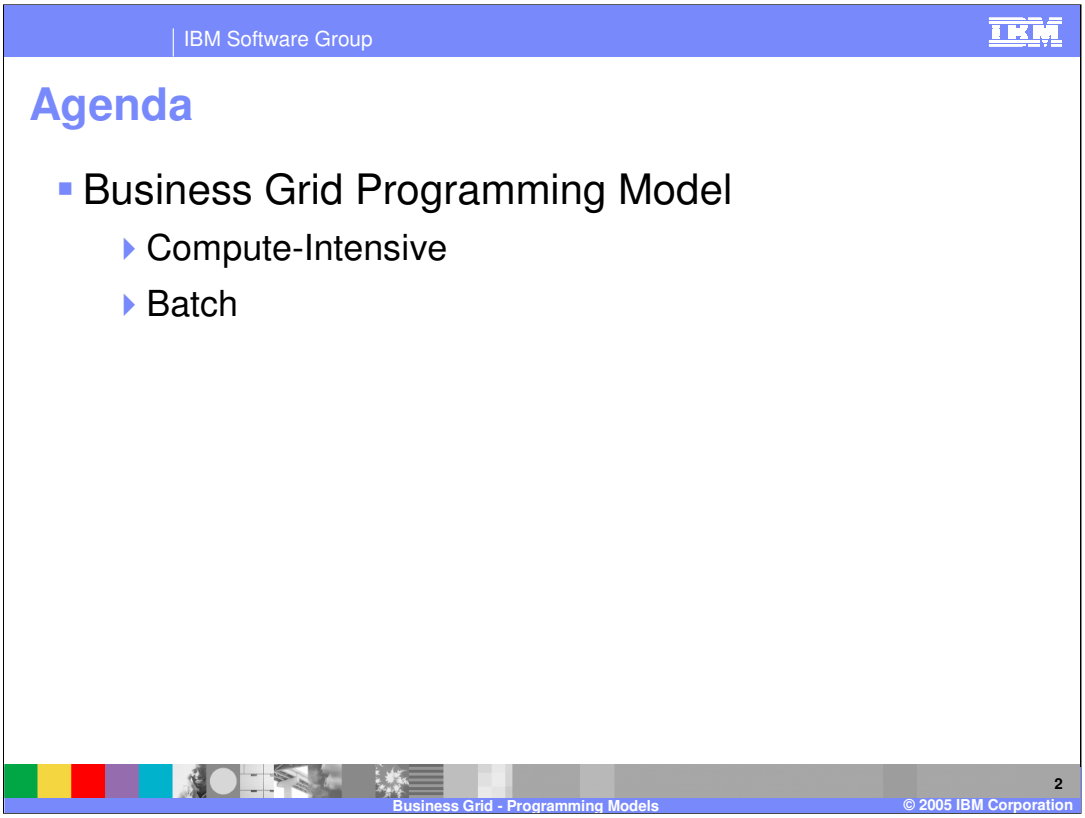


This presentation will provide an explanation of the programming models that should be used with the Business Grid component offered in WebSphere Extended Deployment V6.

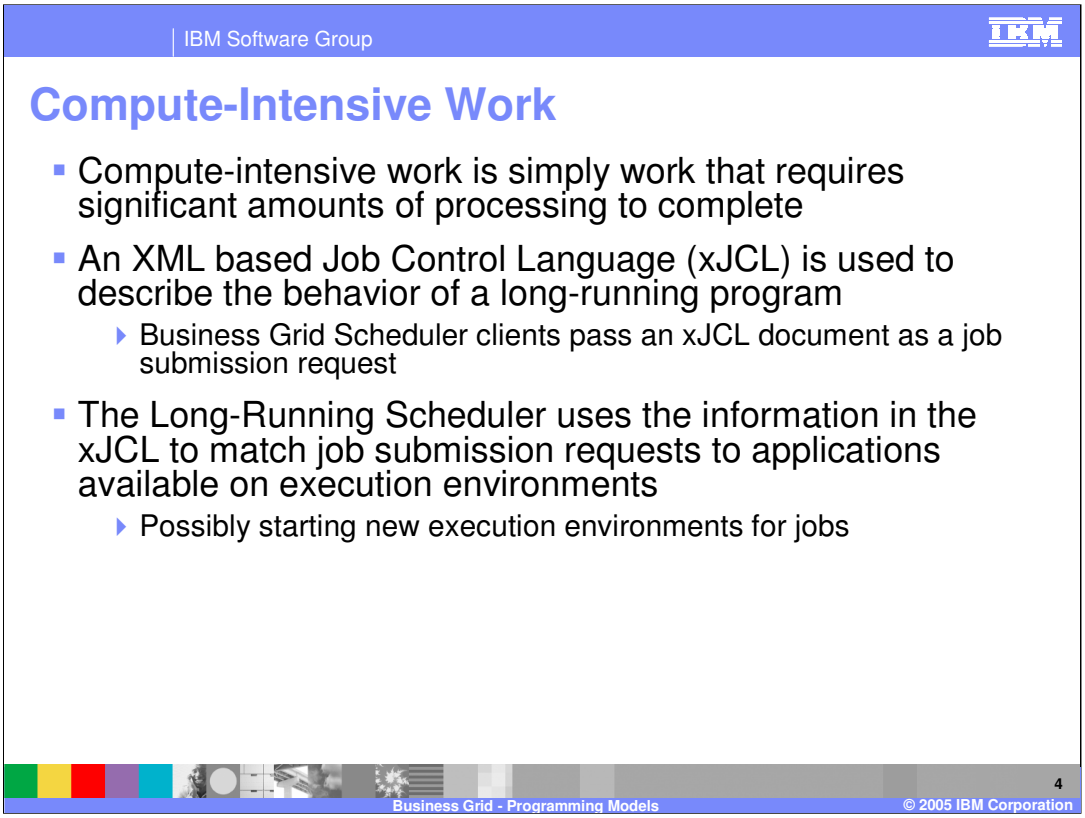


This presentation will explain both the computationally intensive and batch process programming model that are supported by the Business Grid.

Section

Compute-Intensive Programming Model

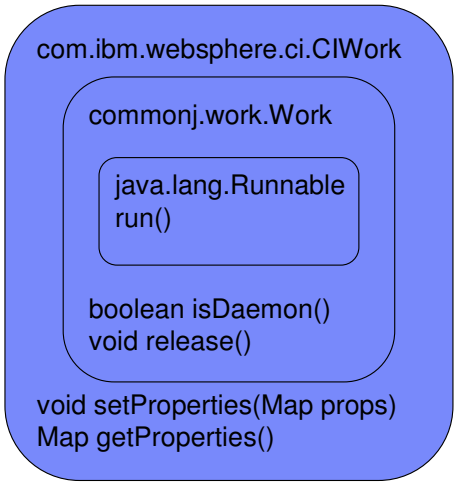
This section will explain the computationally intensive programming model.



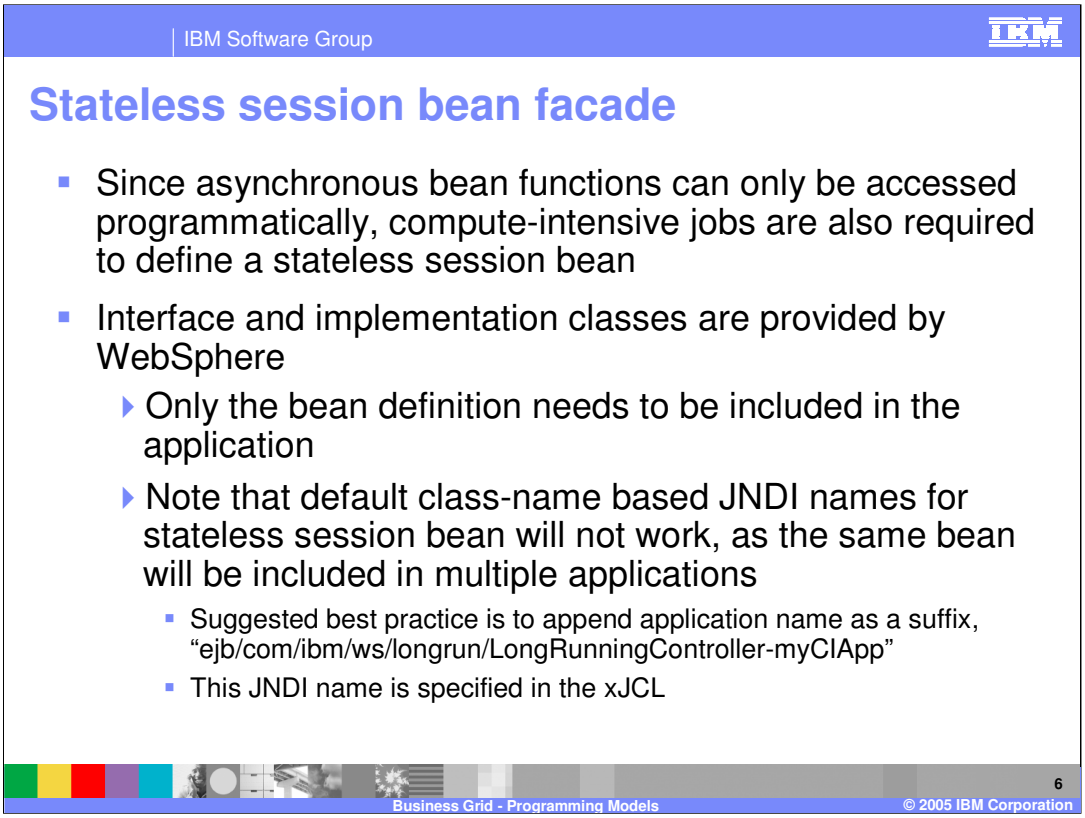
A computationally intensive application is simply an application that requires large amounts of processing to finish. For the business grid, the behavior of a long-running application must be defined within an XML based Job Control Language or xJCL. The scheduler component passes an xJCL document as part of a job submission request. The scheduler uses the information in the xJCL to match the job to available nodes within the environment.

Compute-intensive programming model

- Each job step specifies the name of a class that implements the com.ibm.websphere.ci.CIWork interface
  - A sub-interface of commonj.work.Work
- Additional constraints on classes that implement CIWork:
  - Work.isDaemon() must return true
  - Must have no-argument constructor
  - Strongly encouraged to provide robust implementation of Work.release()

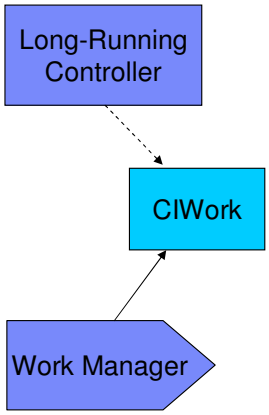


A special interface is used to define the steps of a computationally intensive application. Each step is represented by a class that implements the CIWork interface which is a part of the WebSphere asynchronous bean programming model. Each step's class must have a no-argument constructor, and the Boolean isDaemon() method must return true. Developers are also encouraged to provide an implementation for the release() method, which will be used to remove this step from the long-running environment if a job is cancelled.



Each step of a computationally intensive program is written as an asynchronous bean. Since asynchronous bean functions can only be accessed programmatically, the applications must also define a controller bean, which is a stateless session bean defined in the compute-intensive application's deployment descriptor and allows the execution environment to control jobs for the application. The implementation of this stateless session bean is provided by WebSphere. The application's only responsibility is to include the stateless session bean in the deployment descriptor of one of its enterprise bean modules. Exactly one controller bean must be defined for each compute-intensive application. Since the implementation of the controller bean is provided in the WebSphere runtime, application deployers should not request deployment of enterprise beans during deployment of compute-intensive applications.

Compute-intensive execution environment



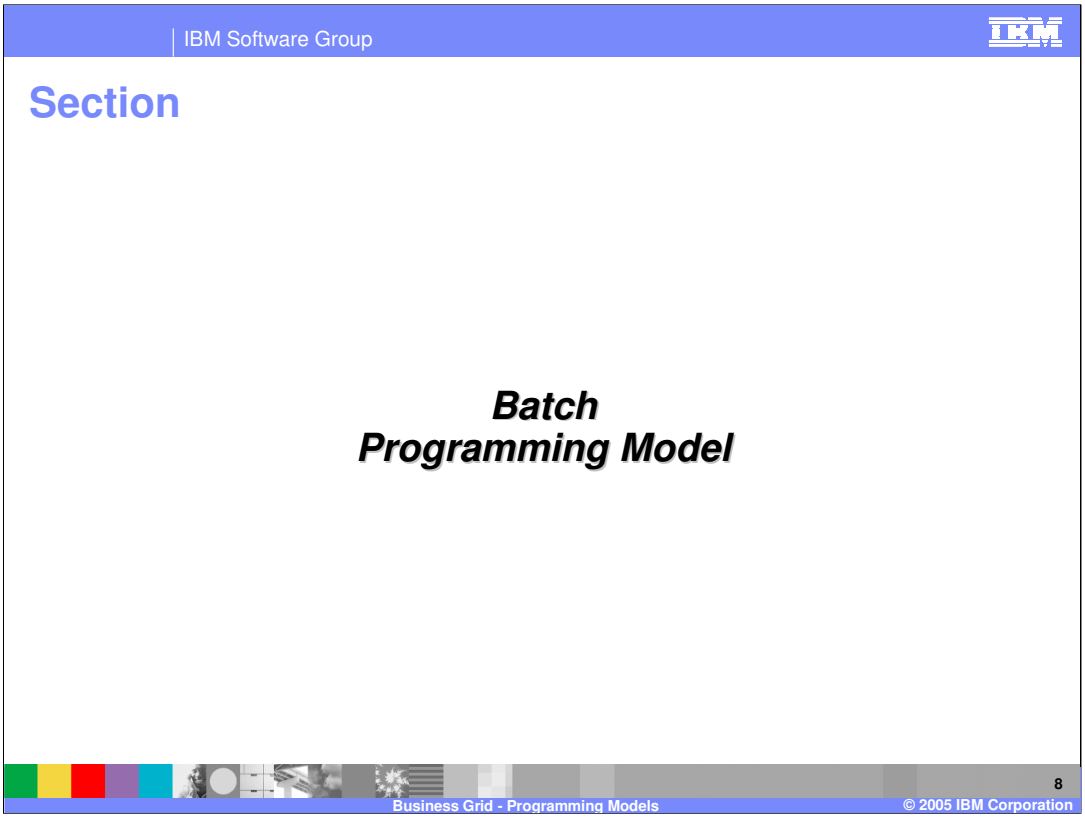
- For a job step, represented as a Stateless Session Bean, the following steps occur
  - The Long-Running controller instantiates a CIWork asynchronous bean
    - Calling the no-argument constructor
  - Sets the properties based on parameters in the xJCL
    - Calling setProperties(Map)
  - Arranges for CIWork to be run by the WorkManager
    - Calling Run() to begin processing
    - Call Release() to cancel the job

A compute-intensive application is started by the application server in exactly the same way as other Java™ 2 Enterprise Edition (J2EE) applications. If the application defines any startup beans, they will be executed when the application server starts. When a job arrives for the application, the compute-intensive execution environment invokes the CIControllerBean stateless session bean defined in the application's EJB module deployment descriptor. The JNDI name of this stateless session bean is specified in the xJCL for the job. For each job step, the CIControllerBean:

- Instantiates the application's CIWork object specified by the class-name element in the xJCL for the job step using the CIWork class's no-argument constructor.
- Invokes the CIWork object's setProperties() method to pass any properties defined in the xJCL for the job step.
- Looks up the work manager defined by the enterprise bean module's deployment descriptor and uses it to asynchronously call the CIWork object's run() method.

If the job is cancelled before the run() method returns, the CIControllerBean invokes the CIWork object's release() method on a separate thread. It is up to the developer of the long-running application to arrange for logic in the release() method to cause the run() method to return promptly. The job will remain in a *cancel pending* state until the run() method returns.

If the job is not cancelled and the run() method returns without throwing an exception, the job is deemed to have completed successfully. If the run() method throws an exception, the job will be marked as *execution failed*. Once the run() method returns (either normally or by throwing an exception), no further calls are made to the CIWork object and all references to it are dropped.

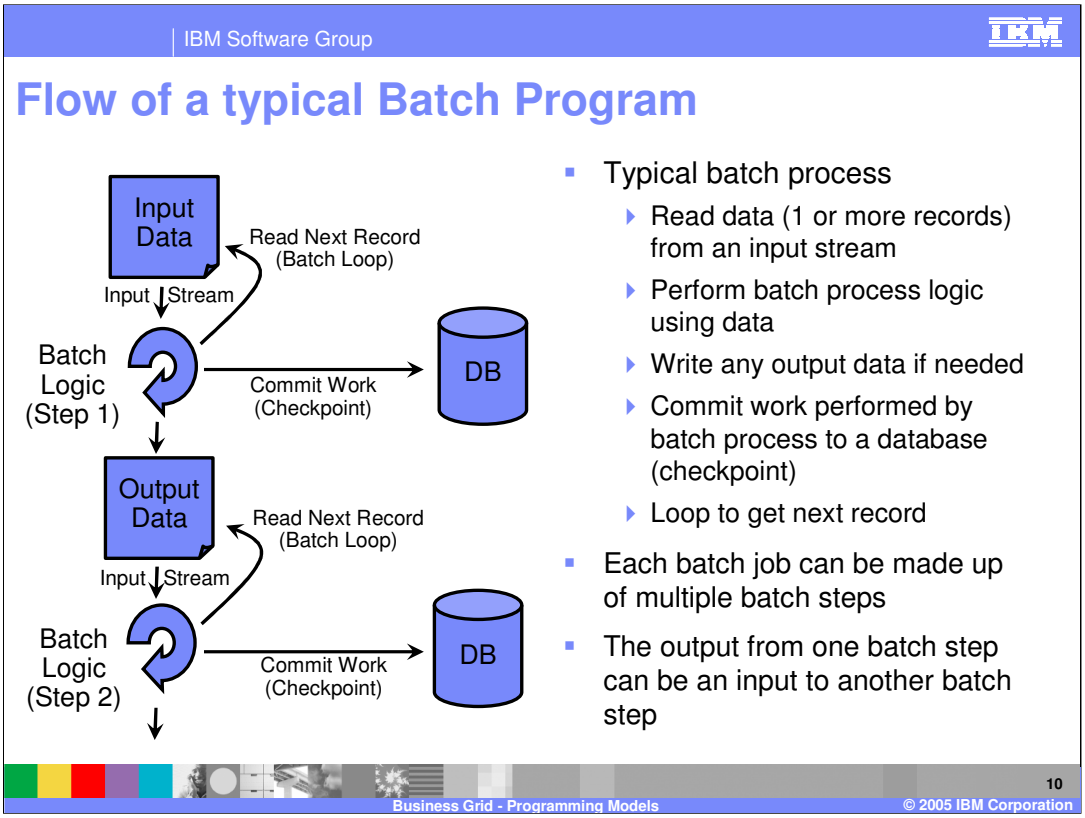


This section will explain the batch process programming model.

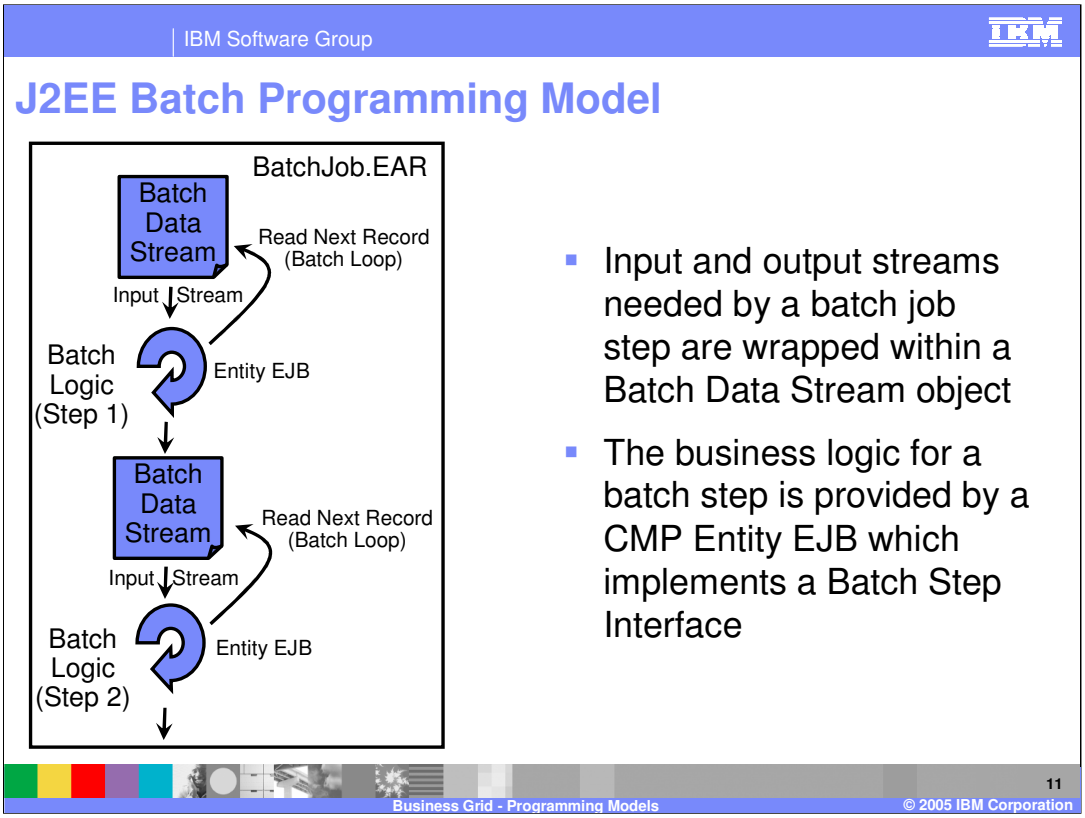
Batch Processing in WebSphere

- Supports a J2EE-based batch processing programming model
- Supports the re-use of existing J2EE services and artifacts in “batch mode”
- Makes it possible to convert legacy batch processes, like CICS® batch programs, to J2EE based programs that are effectively managed by WebSphere
- xJCL is used to describe the behavior of a long-running Batch program
  - Similar to compute-intensive work

Batch applications are Enterprise JavaBeans (EJB) based J2EE applications. Batch applications follow a few well-defined interfaces that allow the batch execution environment to manage the batch jobs for the application. The business grid support for batch application allows legacy batch applications to be appropriately converted to J2EE applications that can be managed by WebSphere. Just like the computationally intensive programming model, xJCL is used to describe the behavior of the job to the environment and is submitted as part of the job.

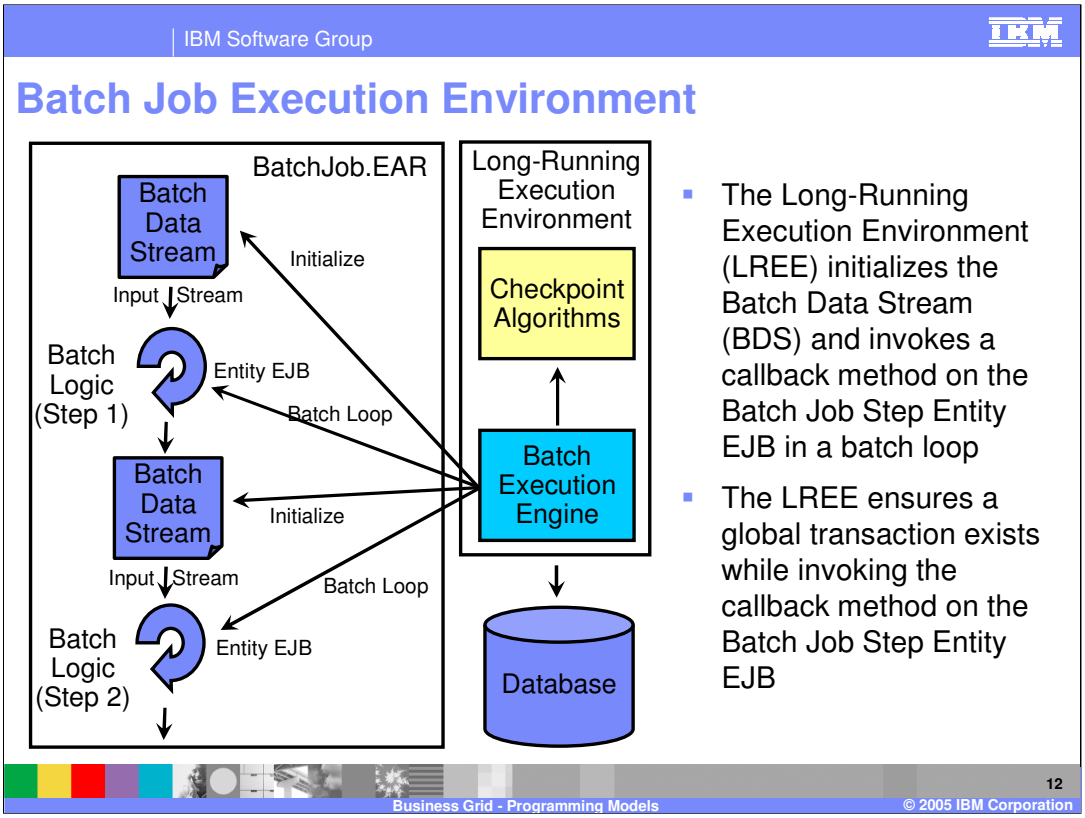


A batch job can be comprised of one or more batch steps. Dividing a batch application into steps allows for separation of distinct tasks in a batch application. In a typical batch process, the application reads data from an input stream, performs business logic on that data, writes output data if needed, commits the work to a database, and then loops to the next record to repeat the process. The output from one batch step can be the input into another batch step in the process. Each batch job can be made up of any number of individual batch steps.



A batch step is implemented as a local container-managed persistence entity bean that uses the WebSphere provided home, business and key interfaces. The business interface, com.ibm.websphere.batch.BatchJobStepLocalInterface, of a batch step EJB provides methods that the batch execution environment invokes to control a batch application. A batch step can have zero or more batch data streams associated with it. A batch data stream (BDS) is a java class that implements the com.ibm.websphere.batch.BatchDataStream interface. A BDS is a Java object that reads the input stream that contains the data to be processed by a batch step. A BDS can also be an output stream that writes data instead of reading it.

Methods on the BatchDataStream interface allow the batch execution environment to manage the data stream being used by a batch step. For example, one of the methods retrieves current cursor information from the stream to keep track of how much data has been processed by the batch step.



The long-running execution environment uses checkpoint algorithms to decide how often to commit global transactions under which batch steps are invoked. The xJCL definition of a batch job defines the checkpoint algorithms to be used. In its deployment descriptor, a batch application is required to declare a special stateless session bean. This bean acts as a batch job controller and must contain enterprise beans-references to all the batch step enterprise beans being used in the batch application. The implementation of this bean is provided by WebSphere, not by the batch application, and it only needs to be declared in the batch application's deployment descriptor. Only one controller bean can be defined per batch application.

Checkpoint Algorithms

- Checkpoint algorithms control the lifecycle of the global transactions started by the LREE
  - Upon committing the global transaction the LREE retrieves cursor information from the BDS and stores it to the Batch Database
- Checkpoint policies are applied to a batch job when it is submitted
  - These policies determine which checkpoint algorithm to use for a particular batch job
- WebSphere XD V6.0 will contain 2 checkpoint algorithms
  - Time Based
  - Record Based
- A Checkpoint SPI is also provided which allows writing custom Checkpoint algorithms and plug them into a LREE via xJCL

The long-running execution environment uses checkpoint algorithms to decide how often to commit global transactions under which batch steps are invoked. These algorithms are applied to a batch job through the xJCL definition. Properties specified for checkpoint algorithms in xJCL allow for checkpoint behavior, such as transaction timeouts and checkpoint intervals, to be customized for batch steps. WebSphere Extended Deployment provides a time-based checkpoint algorithm and defines an SPI for building additional custom checkpoint algorithms. On each batch step iteration of processJobStep method, the LREE consults the checkpoint algorithm applied to that step to determine if it should commit the global transaction or not. Callback methods on the checkpoint Algorithms allow the LREE to inform the algorithm when a global transaction is committed or started. This enables the algorithm to decide if the time has come to commit the global transaction.

Summary

- WebSphere XD provides an environment for managing and executing batch-style and compute-intensive applications
  - Jobs are scheduled using the Long Running Scheduler (LongRunningScheduler.ear)
  - Jobs are executed in the Long Running Execution Environment (LREE.ear)
- A WebSphere XD Business Grid can dynamically balance the needs of long-running work against the needs of transactional applications within a cell

In summary, this presentation explained the benefits of the business gird provided by WebSphere Extended Deployment V6. It discussed the differences between computationally intensive and batch programs and how to create them using the different programming models.

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a registered trademark of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.

Note to U.S. Government Users - Documentation related to restricted rights-Use, duplication or disclosure is subject to restrictions set forth in GSA ADP Schedule Contract and IBM Corp.