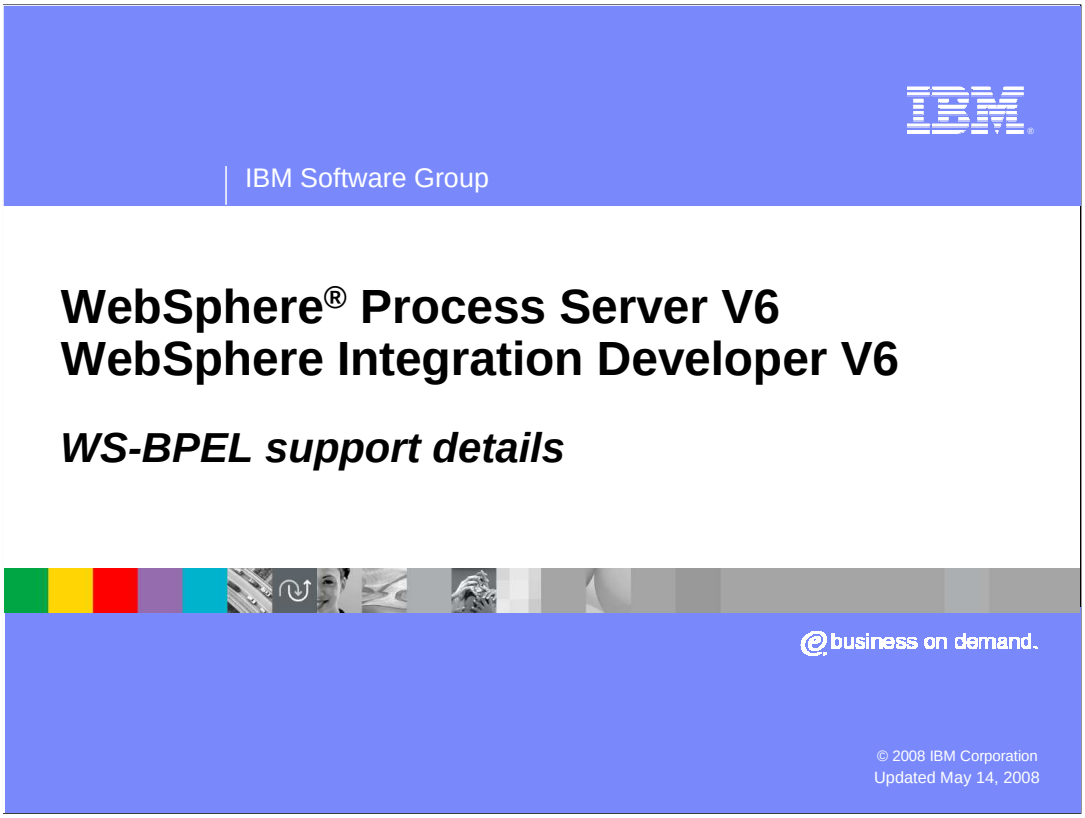


This presentation provides an overview of the WS-BPEL specification and details of the support provided for it in WebSphere Process Server V6 and WebSphere Integration Developer V6.

Goals

- Describe the WS-BPEL support provided by WebSphere Process Server and WebSphere Integration Developer
- Describe the IBM enhancements and additions to the WS-BPEL specification

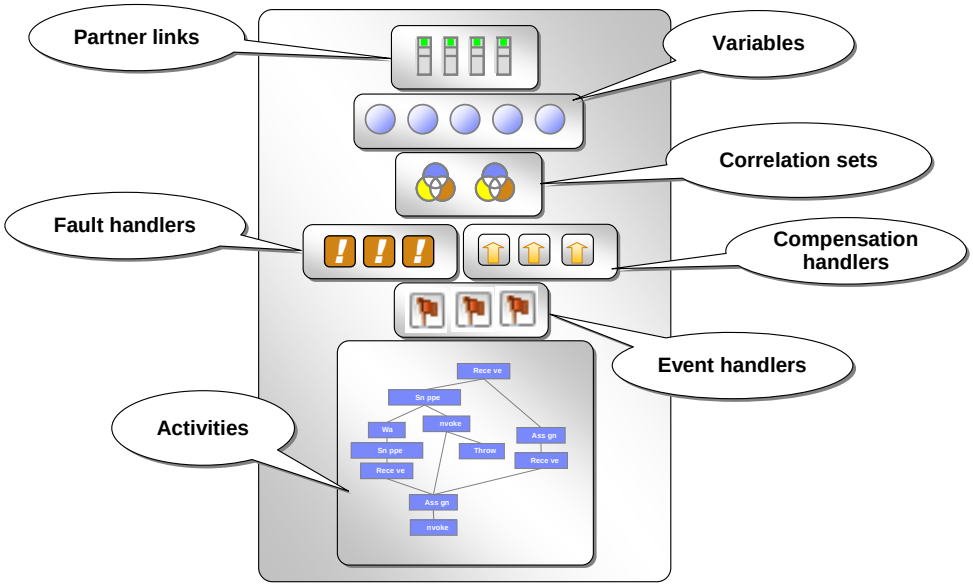
The goal of this presentation is to describe in detail the WS-BPEL support provided by WebSphere Process Server and WebSphere Integration Developer, and the enhancements to the WS-BPEL specification that are supported by IBM.

Agenda

- Details of WS-BPEL support
- Summary

This section of the presentation explains the specifics of WS-BPEL support.

Main elements of a BPEL process

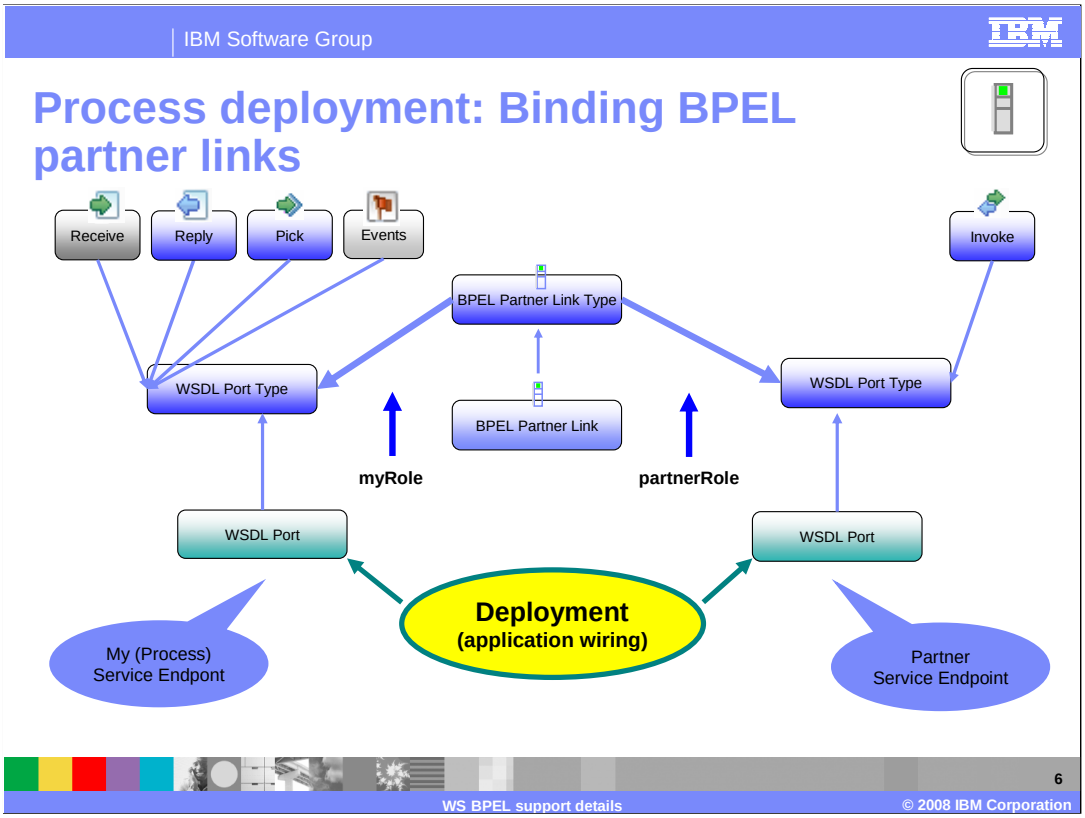


The WS-BPEL specification describes several elements for defining business processes. These elements describe how a business process is organized and how it might be run. The primary elements or concepts are partner links, variables, correlation sets, fault handlers, compensation handlers, event handlers, and activities. Each one of these is covered in more detail in this presentation.

Partner links and roles

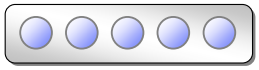
- In business-to-business scenarios, the business process of one business entity needs to interact with the business processes and other services of another business entity
  - WS-BPEL introduces the business partner as a construct in the business process
  - Partner links can be one way or two-way
  - The sending and receiving sides of a partnership are distinguished by the **role** they play
    - for example, buyer/seller, producer/consumer, and so on
- Partner links only supported at the process level

The WS-BPEL specification uses a service-oriented approach to define business processes. The various steps that make up the business process are exposed as services. Within the definition of the business process, there is some type of representation of that service, known as a partner link, which is essentially an endpoint that represents the service you are going to call. The partner link is only defined with the interface of that service. No implementation details are included in the partner link information. Partner links have roles associated with them which are used to define, in an abstract manner, the relationship between the partner link and a particular process. For situations where there are multiple communications between a process and a service, you might have different roles being played. The endpoint or service is identified by a particular role. Partner links are typically one-way relationships; roles really only become significant in two-way relationships. For WebSphere Process Server V6, partner links are only supported at the process level while the WS-BPEL specification outlines support for partner links to be specified at other levels.



BPEL processes use partner links not only to define services that are invoked from the business process, but also to define the interface of the BPEL process. This includes how clients or outside entities contact and interact with the business process. That is why a WSDL port is defined for the invoke action (right side of slide) and another is defined for the receive and reply activities (on the left side of the slide). These WSDL ports are used by clients to send in data and receive a response. Pick activities and events are also used for sending in data and are defined with a service interface. At deployment time, interface information for a particular step is mapped or bound to the actual implementation. Similarly, the protocol (such as Web services, JMS, or SCA) that is used to call the business process is also defined at deployment time. Implementation details are not needed or used within the definition of the BPEL process.

Variables



- Hold data that represents the state of a process
  - The data may be received from or sent to partners, or might never be exchanged with partners
  - Can be specified as input or output variables for invoke, receive, and reply activities
  - Preferably specified using XSD data types

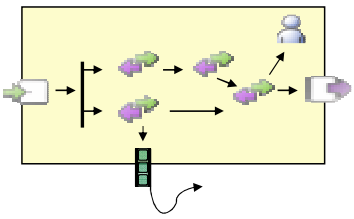
Variables represent the state of the business process. Variables store information about data coming in to the business process that is then reused and sent to the various steps that are part of the business process. They are also used to hold the data that is returned from a service or to hold internal information such as counter values for iterative processes. The variable capability has been enhanced to allow the use of XSD data types, as suggested by the WS-BPEL specification. WSDL message types can also be used to facilitate the service oriented architecture that BPEL builds upon. The business process steps are defined as an interface that has messages and those messages can be used as the basis for your variable types.

Correlation of messages to instances

- The business process execution engine:
  - Accepts an incoming message
  - Correlates the message to a process instance by matching the correlation properties in the correlation set
    - Compares the value of each property to the initialized values in the set
    - If all the properties and all the values match, the message is directed to the appropriate business process instance

Order number

Customer number
Order number



Order number
Customer number
Timestamp

In a long-running process there are points in the process where state data is captured and the process goes into 'waiting' state, waiting for some sort of response, such as a manager approving an expense. To make sure that the approval is associated with the right expense, some unique identifier is used. The unique identifier can be generated and assigned by the system or it can be a combination of business relevant data, such as a timestamp and employee number. Through the lifetime of the process this correlation ID might change. BPEL addresses this with correlation sets. An important point to understand about the correlation ID is that it is something that is known by the outside activity without any notification provided to the outside activity ahead of time. Also, the correlation ID is not something that is generated by the runtime (although the business process container does enforce unique instances). The correlation ID is required at the time the business process instance is started. Typically the correlation ID value is some unique value known to the starter of the business process or is generated before the Business Process is started. When the business process is started, the correlation ID is passed on the input message. Consider a Stock Trading process where many different customers might be looking to buy or sell stocks. Each customer contacts their broker with the intention to trade a particular stock. The broker starts an instance of the Stock Trade process with a unique identifier of customer number. The customer number is unique, and obviously known to the customer. When it comes time to buy or sell a stock, the customer number can be specified on the request and matched to a particular running instance.

Correlation sets

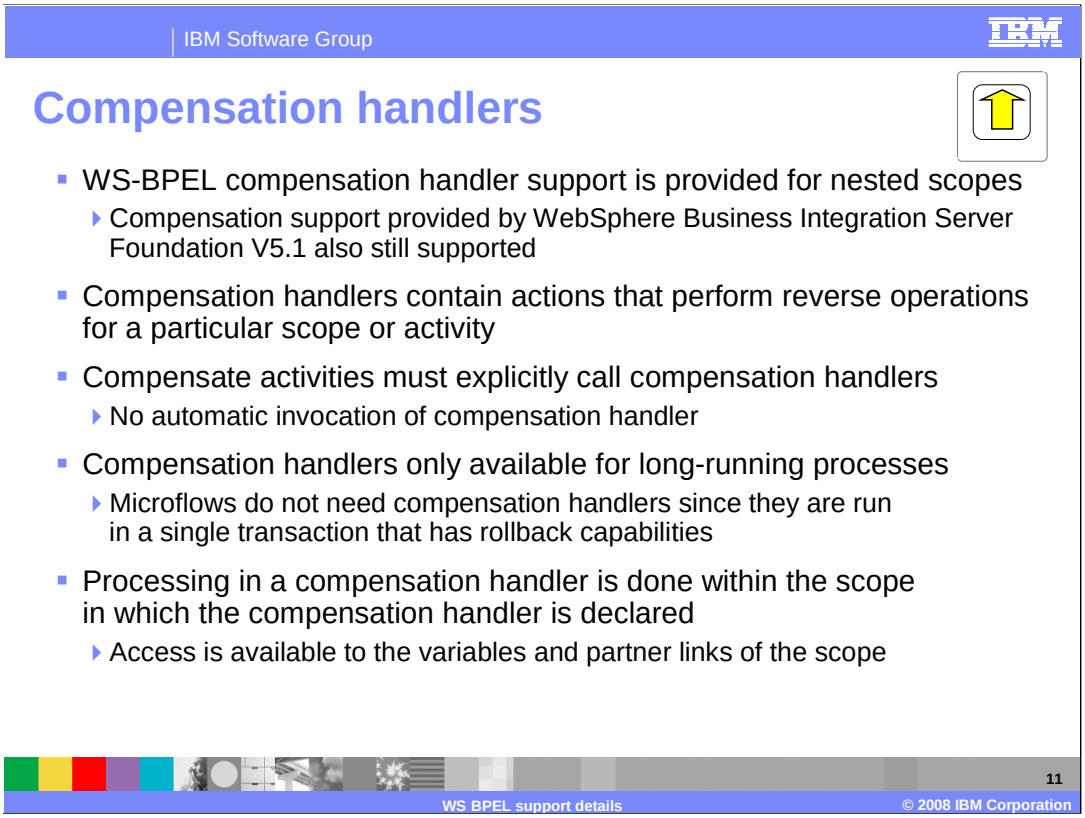
- Correlation sets route an incoming message to the existing business process instance that should handle it
  - Each inbound message must contain a value or set of values that uniquely associates its contents with a specific process instance
    - Examples: customer ID, order number, trade ID, invoice number
- A correlation set has a unique name (called an alias), that:
  - Defines a name for the set of properties
  - Lists the properties to be included in the set
- At run-time, the *values* of the properties in the received message determine the business process instance to which that message is routed
  - The unique values must be initialized the first time the correlation set is used
- Correlation sets are only supported for processes

Correlation sets match incoming requests to the correct business process instance. For example, if there are 1000 loan applications in progress and the credit department is sending in credit report messages for multiple applications, correlation sets ensure that the message is sent to the correct instance. Within a correlation set, you will define the parts of an incoming message that indicate uniqueness. These will then map to some value in the business process state. The messages can then be associated with the instance that contains the same unique values. Note that correlation sets are constructed from application data. The business process must be designed using some value that is unique within the enterprise. Examples include a social security number, order number, customer ID or some other value that is defined as part of the business process. Correlation sets can consist of a single value or multiple values, and there can be multiple correlation sets for long-running business processes. As the business process state changes, there might be different unique identifiers that must be used to identify an instance. The correlation set must be initialized, typically at the beginning of a long-running business process, if there are to be any inbound messages coming into that process later in the cycle. WebSphere Process Server V6 supports correlation sets at the process level, even though the WS-BPEL specification supports correlations sets at other levels.

Fault handlers

- Every fault handler is associated with a scope
  - Handles faults thrown in its scope
- A scope can have multiple fault handlers
  - Different kinds of faults can have different fault-handling activities
- The handler specifies the activities to be performed when a fault has been thrown
- A fault reaching a fault handler means that the remaining processing in the current scope cannot continue
  - All active work within the scope is terminated

Fault handlers are designed to detect and signal a failure in the running of a business process. They are designed to catch primarily business exceptions such as inventory falling below a set limit or a customer requesting an item that is no longer stocked. The fault handler can catch these types of failures and perform an operation to either handle the exception and continue processing or generate a failure to be handled in some other way. A fault handler is associated with an activity or with a scope that encompasses a set of activities. When a fault is caught by a handler, the activity is ended and control is transferred to the fault handler. Fault handlers can be nested similar to the way that scopes are nested. Scopes can contain a set of activities and one scope can contain another scope (or scopes) and each of these scopes can in turn have its own fault handler. If a fault is not caught at the current scope, it will propagate up to the next scope and be caught by the fault handler for that scope. If the fault is not handled at that level, processing continues up the hierarchy. If the fault reaches the upper-most scope of the business process and is not handled, that indicates a failure of the entire business process. In this case, the client or some other fault monitoring tool is expected to handle the fault. A fault handler can also catch runtime failures. However, it is better not to catch specific runtime problems because that exposes implementation details in the business process.

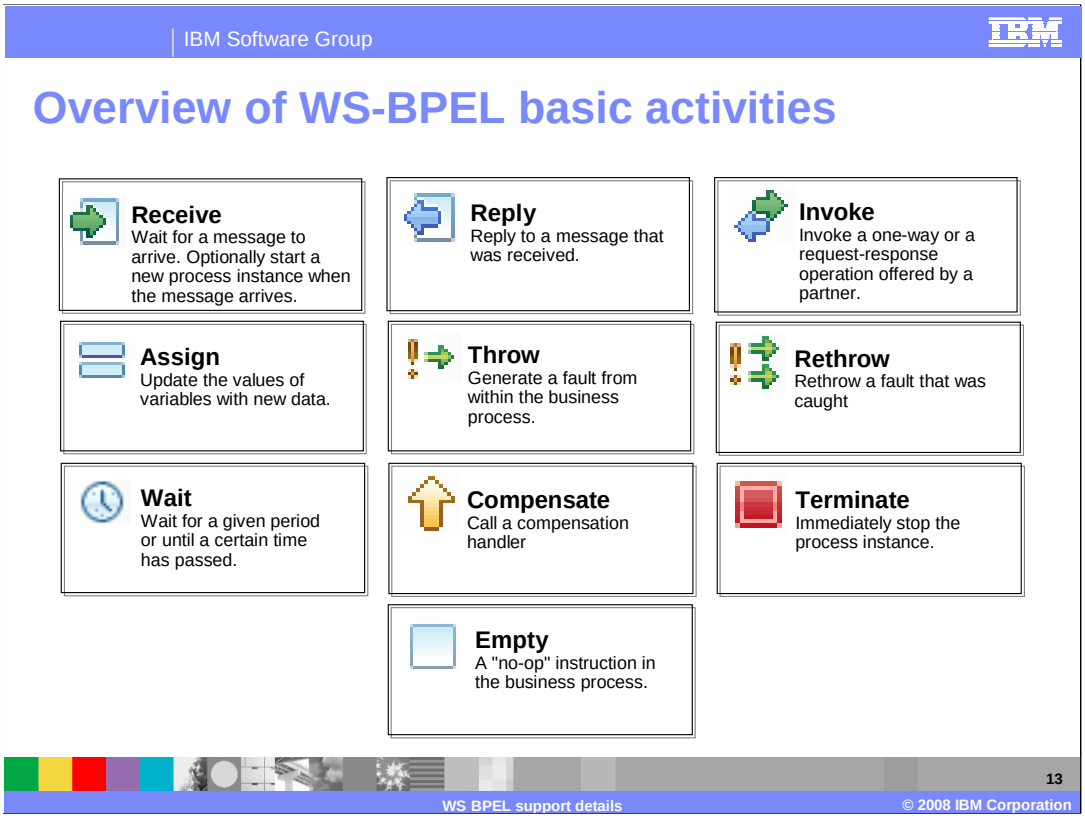


WebSphere Process Server and WebSphere Integration Developer both include support for using compensation handlers as defined in the WS-BPEL specification, in addition to the compensation support initially provided by WebSphere Business Integration Server Foundation V5.1. Back then, IBM had a compensation strategy of its own, and that strategy is still supported in V6. However, WS-BPEL compensation is now supported and that should be used rather than the initial IBM implementation whenever possible. “Compensation” means to perform some sort of opposite or reverse operation to undo something that has been completed. Because short-running, non-interruptible business processes consist of transactions that can be rolled back in the event of failure, compensation handlers only apply to long-running business processes consisting of multiple activities. However, compensation can be useful even within a microflow for reversing the effects of invoking services that are not under transactional control. The roll back capability can be used during compensation as part of an undo operation, and the IBM compensator extends this capability to allow for roll back of short-running non-interruptible processes that do not have transaction awareness. Compensation handlers provide roll back capability for long-running processes and include all the steps that must be taken to roll back the completed portion of a process. For example, if a loan application process includes some sort of warning based on information from a credit agency, it might be necessary to begin undoing some of the steps that have already been completed. Some of the steps might include notifying the office where the application originated that the loan process has been cancelled. Compensation handlers can only be invoked by a compensate activity, which is discussed later. All the variables and partner links that are defined for a particular scope are available during compensation so that the process can be restored to its original state. In WebSphere Process Server V6, compensation handlers are only available for nested scopes while the WS-BPEL specification outlines other areas where compensation handlers can be used.

Event handlers

- Event handlers accept outside requests for a particular scope or for the entire process
  - Requests can be one-way or request/response
- Correlation sets must be enabled on event handlers to direct events to the correct business process instance
- Any type of processing can occur within an event handler
- Alarm event handlers are available to perform actions after some time period has expired
  - Timer starts when scope is entered

Support for event handlers, originally defined in the WS-BPEL 1.1 specification, is provided in WebSphere Process Server V6. Event handlers accept requests from external clients into the business process. Event handlers are defined at a particular scope of activity and events can be accepted during the entire cycle of the business process, or they can be limited to a specific scope. Event handlers use correlation to provide an interface to outside entities and to match up requests to the correct process instance. When an event is received by a handler, a set of steps defined in the handler is run on a separate thread from the business process itself. The event handler does have access to business process state information such as variables and partner links. When the event handler activities are complete, the thread goes away. Alarm events can also be set up within an event handler. Alarm events are triggered after a predefined period of time, and the timer begins when the scope containing the event handler is reached. Event handlers are only available for long-running business processes.



Here is an overview of the basic BPEL activities that are used primarily to carry out the different steps in the business process. The Receive activity is an asynchronous activity that can be added to a business process. The Reply activity is used in conjunction with a Receive activity when a request/response is used by an outside action to communicate with the business process. Invoke activities are the main activities that interact with outside entities and service providers. Assign activities are the primary way that data is transformed and moved from one variable to another. Throw activities are for indicating some type of failure has occurred within the business process and it needs to either be handled in the business process or passed back to the client because no further processing can occur. If a failure is caught, a Rethrow activity can be used to throw the same failure without needing to use an Assign activity to move the failure contents from one failure to another. The Wait activity allows for a business process to stop and wait for a specific amount of time. The Terminate activity can be used in a business process where all processing should end with no way of compensating or performing reverse processing on the instance. The Empty activity acts as a placeholder in a business process for a subsequent activity that might be implemented. When the Empty activity is reached, the business process continues without stopping, treating the activity as a no op. The Empty activity can be changed later to any of the other activity types. Each of these is covered in detail on the next few slides.

Receive and reply activities

- Receive
  - Receive a message sent to a business process
    - Starts a new business process
    - Restarts an existing process with Correlation
  - Message can be request/response or one-way

- Reply
  - Reply to a message that was received
  - Only available as a response to a request/response
  - Can return either
    - response message
    - fault message

The support for Receive and Reply activities has not changed from WebSphere Business Integration Server Foundation V5.1. Receive is a means of sending information into a business process and is typically used at the beginning of the process to accept data and start the instance. A correlation set can be specified, establishing a unique instance from information in the message. Reply activities can only be used in conjunction with a Receive activity and can return a response message indicating success or a fault message. Receive and Reply activities can be used in long-running interruptible or in short-running non-interruptible business processes. Even in a long-running process with a synchronous interface, there are asynchronous implementations, such as JMS, that can be used to carry out request/response types of interactions.

Invoke activity

- Invoke
  - Invoke a one-way or a request/response operation offered by a partner
  - Call another business process
- IBM extensions to Invoke activity
  - transactionalBehavior – explicit checkpointing
  - continueOnError – enter a stopped state after infrastructure fault
  - expiration – Java™ timeout expression

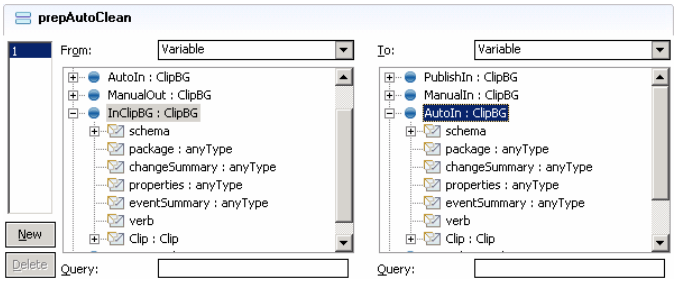
The invoke activity is used to perform the various steps of a business process. This activity calls out to services by means of the partner links, using services or request/response calls. The invoke activity can be used with a long-running service or communications that use JMS queues. It is not necessary to differentiate between a call to a synchronous or an asynchronous activity within the business process. The business process engine can handle this for you by looking at how the invoke and partner link are bound to the implementation. There are several IBM extensions to the Invoke such as the transactional behavior of a long-running process. Transactional behavior determines whether a transaction should be committed around this invoke, or if it should run in its own transaction, or if it should participate in an existing transaction. In some cases, grouping invokes or other types of snippets with invokes can result in better performance because a transaction does not have to be committed for each step of the process. The continue on error flag allows the business process engine to continue on in certain situations, even if there is a failure. For example, a publish operation where no partner is reached is not considered a critical failure and should not cause processing to halt. Expiration can also be set for long-running processes invoked by the Invoke activity. If no response is received in a specified amount of time, the business process can raise a failure.

Assign activity

Activity

Assign

- Update the values of variables with new data
- Values can be copied from a source to a destination
- As an alternative, snippets can be used:
  - to build a complex message part
  - when using custom properties for message parts
- XPath 1.0 support



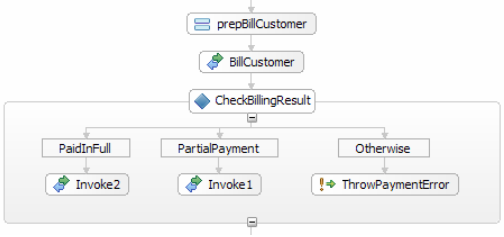
Assign activities are one of the primary means of working with variables defined within the business process. These variables represent the state of the business process in addition to information that is sent to and received from the different services. Variables are based upon business objects, used to define the data types. Different services have different messages that must be populated with data and the assign activity is the primary way to move information from one variable or message to another. Assign activities perform basic mapping of information. For more complex types of mappings, the use of snippets is supported. These can be written in Java or created using a visual editor. There is also a business object map capability in the BPEL editor, which can be used for mapping specific business objects. In WebSphere Process Server Version 6, support for XPath 1.0 is provided, allowing you to specify an XPath query string or to drill down into more complex types of business objects.

Throw

Activity

Throw

- Signals a situation that the immediate logic cannot handle
  - Faults have a name and an optional message, which can contain additional information about the nature of the fault
- May be caught by a fault handler in an enclosing scope
  - The fault name, the optional message, or both can be used in decision making
- Uncaught faults signal a failed business process

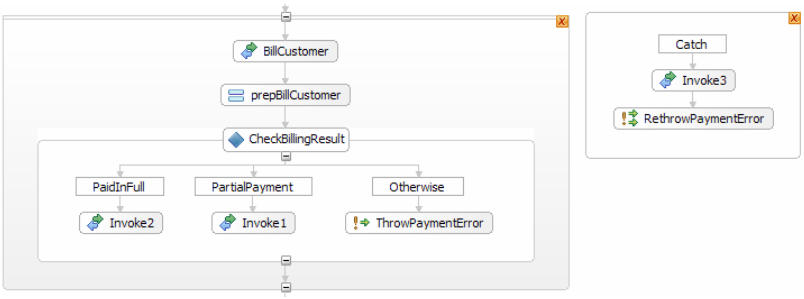


The Throw activity signifies a problem that the business process logic cannot handle. A fault is thrown within the business process and a signal that there is a problem is sent to a higher level in the business process or an outside entity. There are several built-in fault messages that can be used with this activity. Alternatively, you can create custom faults that can be thrown to the next higher scope or the scope enclosing the Throw activity (if it contains a fault handler). If the fault is not caught and handled before it reaches the uppermost level, this signifies a business process failure.

Rethrow

Activity

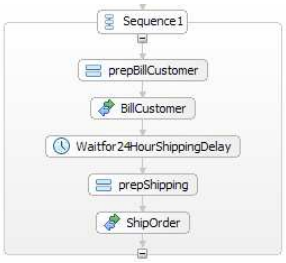
- Signals an error state still exists
- Only available in fault handlers
- Raises new fault with identical name and message as caught fault



The Rethrow activity signals that a previously existing error state still exists. With a Rethrow activity, if a fault is caught, it is rethrown to the next higher scope. In the BillCustomer example, if the CheckBillingResult returns PaidInFull or PartialPayment, processing continues. If the customer has overpaid or has not paid and neither of these conditions are met, that signals a failure and a fault is thrown signifying the situation cannot be handled using the current logic. The catch block then catches the specific fault, allowing notification to be made and the account to be updated in some way. However, instead of issuing a new fault, the initial fault is reused and passed on, simplifying fault notification. In WebSphere Business Integration Server Foundation V5.1, you had to use an assign or a new throw activity and move the data from the caught fault into the new fault message, requiring more custom programming.

Wait activity

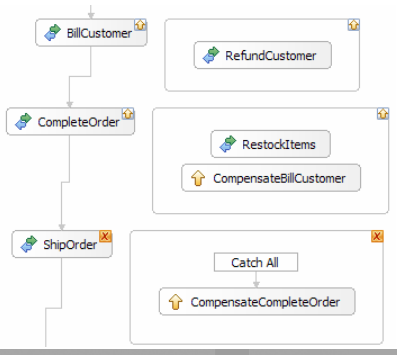
- A **wait** activity stops the business process for a specific amount of time:
  - Duration in seconds
  - By calendar date
  - By XPath expression
  - By Java expression (IBM extension)
- Only available for long-running business processes



The wait activity can be used to stop a business process on the current processing path for a specific period of time or until a specific date has been reached. The duration or date can be either a hard coded value or calculated dynamically. XPath support can be used or a more complex calculation can be performed using Java. The latter capability is an IBM extension. Any length of time can be used for a wait activity, therefore it is only available for use with long-running interruptible processes.

Compensate

- Invoke a reverse operation on an activity or a scope encompassing activity
- Invokes compensation handler of an invoke activity or scope
- Only scopes or activities that have completed normally can be compensated
- Compensate activity only available in a fault handler or compensation handler



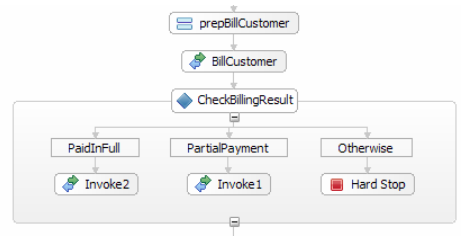
The compensate activity is defined in the WS-BPEL specification and is supported in WebSphere Process Server V6. This activity signifies that an activity or scope of activities that has completed must be undone. This occurs when a failure situation is reached; the compensate activity can only be used within a fault handler or within another compensation handler. The compensate activity points to a specific compensation handler, which contains the steps necessary to undo the completed action. For example, if an order fails to ship, an error is returned. The failure, signaling that there is a problem that requires some processing to be undone, can be caught using catch all (no need to catch the specific fault). The compensation activity is set to point to the appropriate compensation handler and the logic in that handler takes over, processing the activities in the handler, and the items are restocked. The next compensation handler is then called, which handles refunding the customer or crediting their account. This compensation handler has access to all the partner links in the business process and to the variables that were set before the action began. Using these variables, the compensation handler can undo the completed activities or any other processing required to return the instance of the Business Process to the original state. In the example on the slide, although the compensation handlers in this example are defined on invoke activities, they can alternatively be defined on a scope to include multiple activities.

Terminate activity

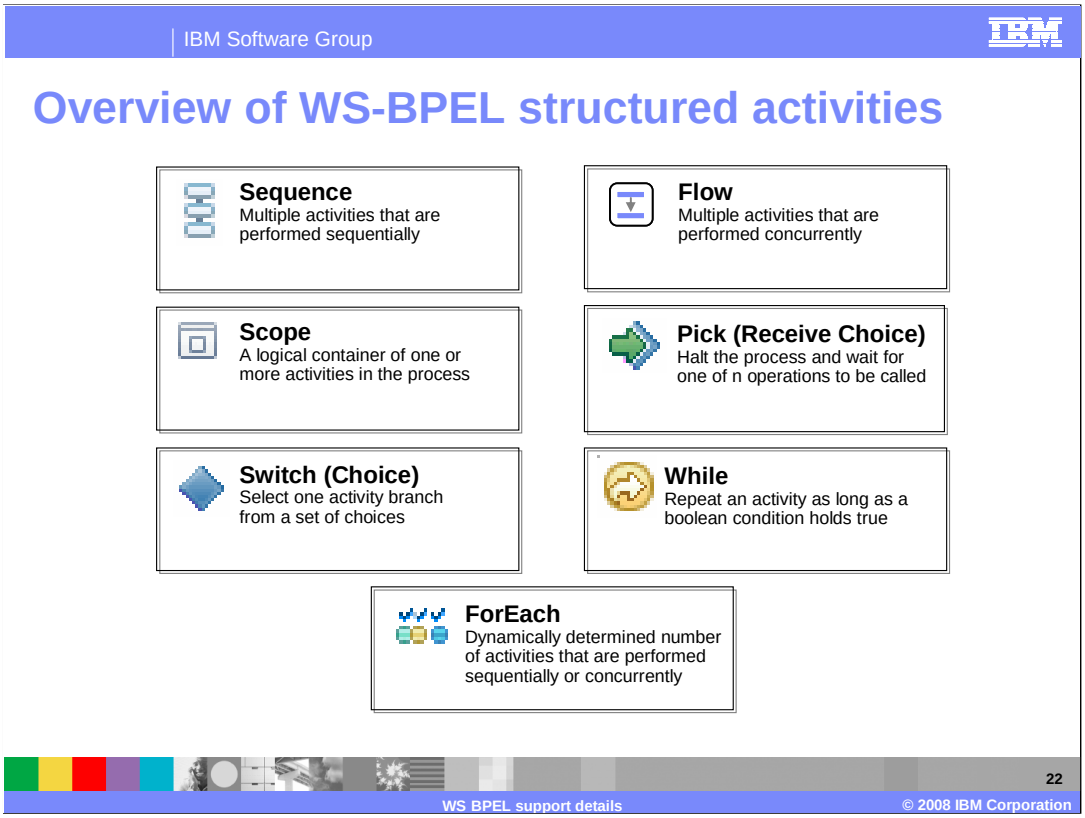
Activity

Terminate

- Terminate activity ends processing of business process instance immediately (hard stop)
  - All processing ends, including parallel processing paths
- Recovery (compensation) of terminated processes is unavailable



The terminate activity is the equivalent of pulling the plug on an electrical device. This is a hard stop that signifies a failure has occurred or the business process should immediately be halted at this point. If a process is terminated, no recovery is possible and no compensation is available.



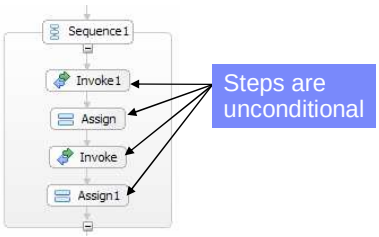
Here is an overview of structured activities, which are used to perform structured programming within a business process. Activities can be run one after another using the Sequence activity, or they can be run in parallel with the Flow activity. A Scope activity defines a collection of activities with their own set of variables, fault handlers, a compensation handler, and event handlers. The Pick activity, also known as Receive Choice, is similar to a Receive activity in that the business process will stop and wait at the activity until a message arrives. In this case, there are several potential operations, and when a matching request arrives, that branch of the Pick is run. The Switch activity, also known as the Choice activity, provides a way to perform conditional logic in a business process. The While activity is a way to repeat a group of activities based on a boolean condition that is evaluated before running each iteration. Finally, the ForEach activity provides a way to perform a dynamically determined number of activities either sequentially or concurrently.

Sequences and flows

Activity

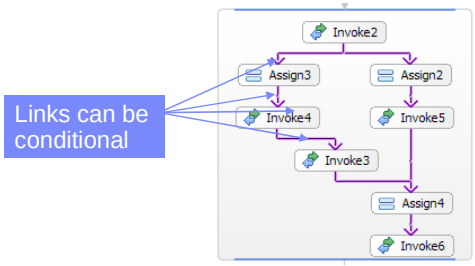
Sequence

- A **sequence** ensures a certain order of processing for nested activities
  - A single activity is run at a time

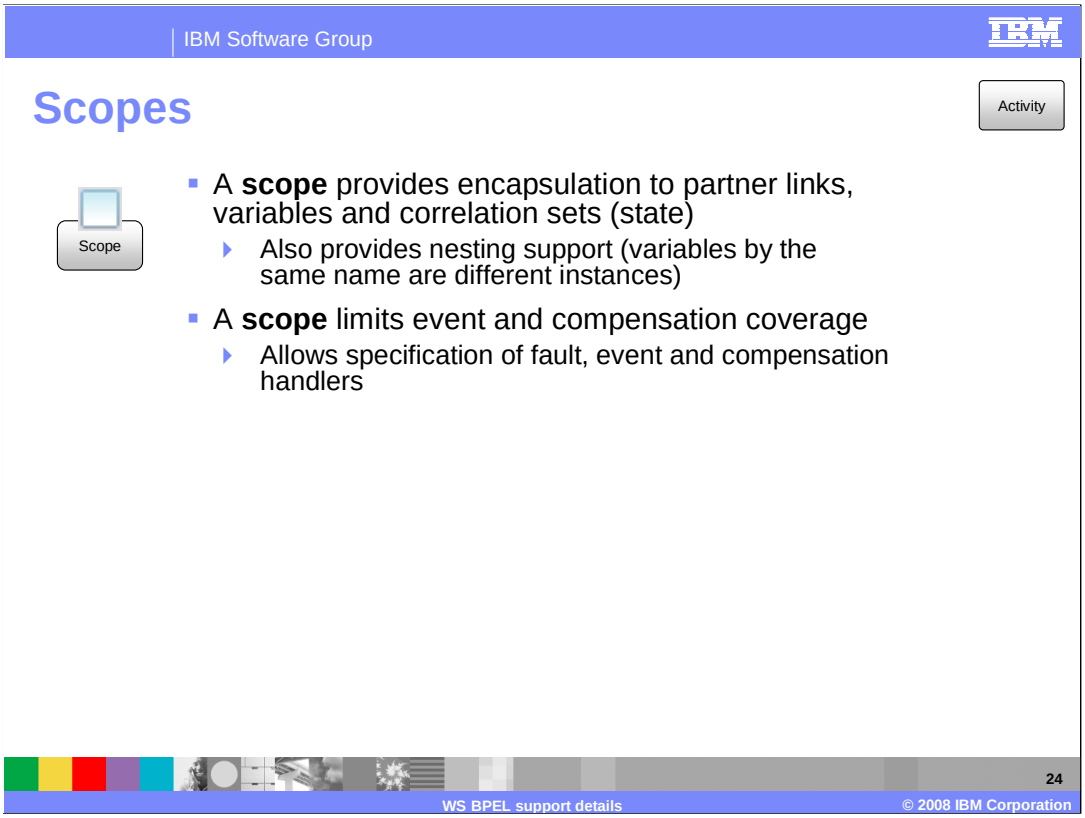


Flow

- A **flow** allows parallel processing
  - Conditional logic can be specified on inter-activity links
  - Flow activities in an uninterruptible process are run sequentially



Shown here are the sequence and flow activities. A sequence consists of activities occurring one after another. Each activity must complete successfully before continuing on to the next one. A flow allows a business process to have parallel processing of activities. Within the parallel path, there are links specified. These links can include logic to check whether processing should continue. This logic can use variable state information from the business process to determine if processing should proceed. Flow activities use multiple threads and are designed to be used with long-running interruptible business processes. In a short-running noninterruptible process, activities are processed sequentially. Where there is more than one branch that can be taken, one branch is selected at random.



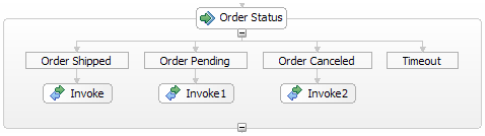
Scopes are supported in WebSphere Process Server V6 as defined in the WS-BPEL specification. Scopes encapsulate correlation sets and the state of a business process in order to make particular values available to a specific set of activities. Scopes can be established to define event handlers and compensation handlers. Scopes can be nested, allowing variables with the same name to exist in the business process as unique instances. If a variable is defined at a particular scope, that variable is accessed when called by name. Variables defined at a parent scope are available as long as the name is different. Once the scope ends, the variable is no longer available to downstream activities.

In WebSphere Integration Developer, the business process editor shows variables in the so-called “tray” on the right side of the editor. The content of the variables category in the tray depends on the selection in the editor. If the process or an arbitrary activity other than a scope is selected, only global variables, that is, variables that are declared on the process itself, are shown. When a scope with variables is established in the business process editor, and that scope is selected, the business process editor will only show variables defined for that scope. This is a current limitation of the business process editor and is likely to change in the future. However, note that the global variables are still available. For example, if you insert an assign activity inside the scope, you can select from all variables visible to the assign activity, that is, all global and all local variables.

Pick (receive choice) activity

Activity

- A **pick** activity is a combination of a **receive** and a **switch**
- The business process waits for one of n possible messages before continuing
- Correlation set must be specified for each message pick
- Each message pick can have different security permissions
- The OnAlarm property is available to time-out, wait, and continue
  - Time-out in duration (seconds), by specific date, by calendar, or by XPath expression
  - IBM extension

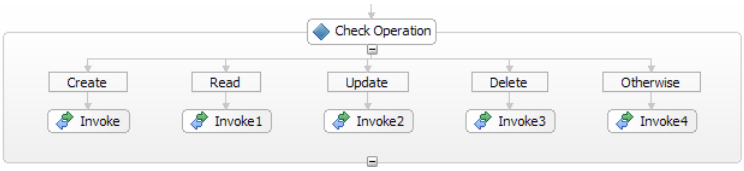


The pick or receive choice activity provides a point within the business process where information can be received from outside entities or from different clients. These points or interfaces can be different in different circumstances. For example, if you are waiting for a message to indicate order status, such as shipped, pending, or cancelled, each status can use a different interface and have different information passed in. When any one of these messages comes into the business process, processing continues down the path of the received message. In the example on the slide, if the order status of pending came in, only the invoke1 activity is invoked. Each of these different interfaces can be secured with different permissions. Order shipped might be coming from the warehouse, whereas order cancelled might be coming from a customer service representative and these activities can be secured in different ways. A timeout capability is also available to prevent a stalled process. Enhancements include support for XPath to calculate the alarm and the timeout value. Timeout values can be hard coded or can be determined dynamically using a calendar, which is an IBM extension.

Switch (choice) activity

Activity

- A **switch** implements a decision chosen from multiple conditions
  - Conditions can be specified as a visual expression, Java expression (IBM extension), XPath, true or false
- The first case condition to evaluate to true is run
- If no case resolves to true, the otherwise case is run
- If the otherwise is omitted, an implied otherwise with condition true is used



A switch or choice activity is similar to a programming case statement where conditions are defined. Whichever condition resolves to true first determines what processing takes place. Conditions are checked from left to right and when one evaluates to true, the activities defined under that condition are run. An otherwise clause can also be used in the case that none of the conditions evaluates to true. If no otherwise clause is specified, it is implicitly generated by the business execution engine as defined under the BPEL specification. Conditions can be specified as a visual expression, Java expression (IBM extension), XPath, true or false.

While activity

Activity

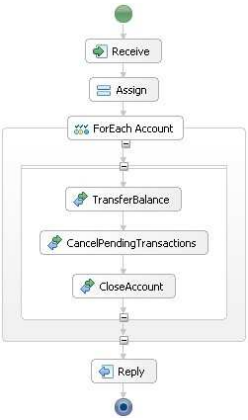
- The **while** activity will repeatedly run the activities in its scope as long as the *condition* is **true**
  - The *condition* is checked before every iteration, so it is possible that *none* of the nested activities are run
- The *condition* can be
  - A visual expression (IBM extension)
  - A Java expression (IBM extension)
  - An XPath expression
  - True or false
- An unhandled fault will terminate the loop
  - It is more efficient to end the loop by meeting the condition than by throwing an exception
- A fault that is handled will also terminate the loop
  - Unless the fault is handled within a scope inside the loop

The while activity is used to run activities iteratively. The activities under the scope are run as long as a defined condition remains true. The condition is evaluated before activity processing, so it is possible that none of the while activities is run. The condition can be specified as either true or false, an XPath expression, or as a visual or Java expression, which are IBM extensions. The while activity is a scope and can have a fault handler associated with it to handle failure of any activity in the while loop that is not caught at the while loop construct level. If a fault does occur and reaches the while scope, any processing in that loop that is running in parallel halts and the fault handler is called. This is not the best way to exit a while loop, so you should try to meet the condition of the while loop and exit that way, because there is additional overhead associated with throwing a fault. The use of faults should be reserved for indicating failures in the business process.

ForEach activity

Activity

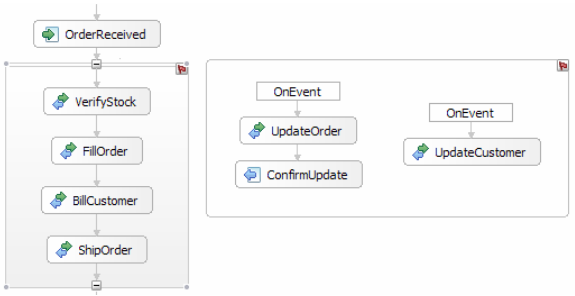
- The **forEach** activity performs a dynamically determined number of activities either sequentially or concurrently
- Performed *sequentially*, the **forEach** activity is equivalent to the “for” loop construct from C/C++ or Java
- Performed *in parallel*, the **forEach** activity is equivalent to a flow activity with *N* similar child activities not constrained by links
- A *completion condition* can be specified if not all iterations or parallel branches are required to complete



The forEach activity provides a way to perform a dynamically determined number of activities either sequentially or concurrently. Performed sequentially, the forEach activity is equivalent to the “for” loop construct from C/C++ or Java, in that the activities contained within the forEach activity are iterated one after the other. Performed in parallel, the forEach activity is equivalent to a flow activity with *N* similar child activities not constrained by links. The set of child activities is referred to as a “branch”. Running the forEach activity results in *N* branches being performed in parallel. The number of sequential iterations or parallel branches is maintained by a counter and is determined by subtracting the counter’s defined start value from its defined final value. The counter’s start value and final value can be specified statically using integer values, or dynamically using either an array and its boundaries or an expression. The expression can be specified as a visual expression, a Java expression, or an XPath expression. A completion condition can be specified if not all sequential iterations or parallel branches are required to complete. The completion condition can be specified statically using an integer value, or dynamically using an integer expression. In the sequential case, once the number of completed iterations reaches the value specified by the completion condition, no further iterations are performed. In the parallel case, once the number of completed branches reaches the value specified by the completion condition, all remaining active branches are terminated. If the successfulBranchesOnly attribute is set to “yes” then only the number of successfully completed branches or iterations is considered.

Event handlers

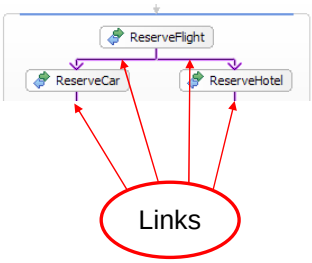
- An **event handler** provides an active receive for a scope
  - However, unlike a receive activity, any number of event handler instances can run in parallel
- May be repeated multiple times until scope is completed
- Can set restrictions on who sends events



Events are associated with a scope. When the scope is started, events can be received until the scope is completed. The example on this slide lists four activities that are grouped into a scope with an event handler defined on the scope. Because these four activities are run at any time, different events can be received. There are two events that can come in while these four activities are run. Each one of those events initiates a different set of activities. The event can be a one-way operation such as that on the right or a request/response like the event handler on the left. Each event handler must have a correlation set specified in order to insure an incoming message reaches the correct process instance. A separate processing thread with access to partner links and variables within the scope is used for each event. Events can be secured differently, allowing only authorized users to call an event. If an event comes in before the event handler becomes available when the scope is reached, the business process engine will hold that event and wait for the scope to be reached. If the event comes in after the scope is ended, a runtime exception is generated indicating that an instance is not available for the event.

Links and transition conditions

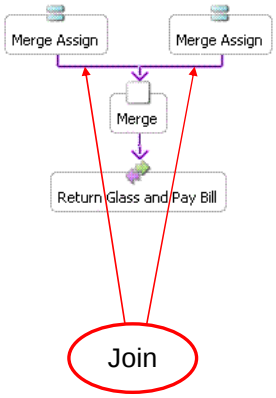
- Links are conditional processing paths *leaving* an activity
- The boolean condition (true, false, otherwise) is derived from:
  - Visual expression (IBM extension)
  - Java expression (IBM extension)
  - Simple (true, false)
  - XPATH
- If multiple links in a **flow** activity evaluate to **true,** and the activity is in a long-running process, then the activities can be run in parallel



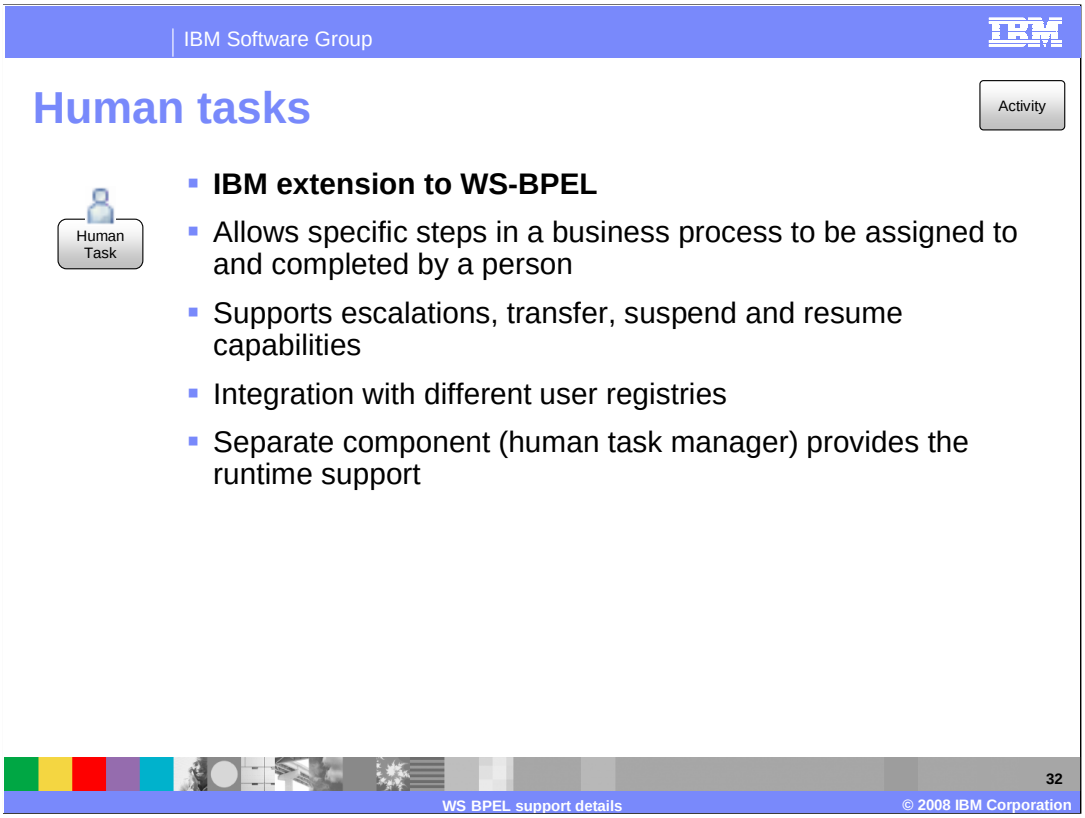
Links and link conditions are associated with flow activities and can include conditional logic to determine if processing should proceed on a particular path. Conditions can be specified as a visual expression (an IBM extension), Java expression (also an IBM extension), XPath, or a simple true or false. Parallel processing can be performed during a long-running business process if multiple conditions evaluate to true.

Join conditions

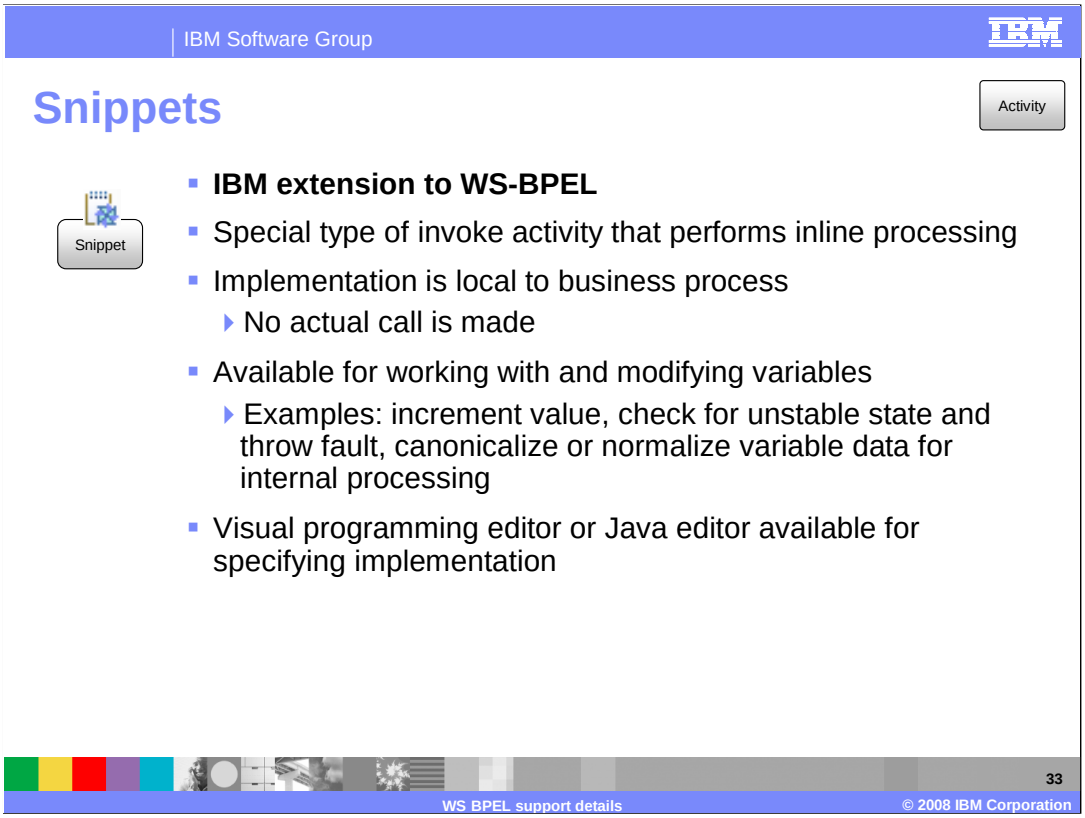
- Joins are conditional processing paths *entering* an activity
- Join conditions are specified on the target activity
- Join conditions can be specified using:
  - Visual expression (IBM extension)
  - Java expression (IBM extension)
  - Simple expression (IBM extension)
  - XPath
- A join condition can be:
  - **Any –** logical *or* (this is the default)
  - **All –** logical *and*
- Control flow enters the activity following a join
  - If the join condition is **all** and *all* of the links are **true**
  - If the join condition is **any** and *one* of the links is **true**
- If none of the links evaluate to **true**, the activity is skipped and dead path elimination occurs



A join is when there is more than one path entering an activity. A join condition is logic that indicates if the process should wait for processing from all paths or just a single path before continuing to run an activity in a flow. In some situations, it might be appropriate to create different paths, but it is not critical that both paths complete before continuing on with processing. Join conditions can be used to accomplish this. Join conditions can be specified using a visual expression (an IBM extension), Java expression (also an IBM extension), XPath, or a simple true or false. For the join condition, there are two settings. A value of “All” requires all links to be true before continuing, and a value of “Any” only requires that one link evaluates to true before continuing. If there is a single link that resolves to false or one of the links resolves to false with a join condition setting of “All”, then Dead Path Elimination is used. The activity is skipped and any links coming out of the activity will have a negative value. This allows for the business process to continue, but not run incorrect activities.



Human tasks are an IBM extension to the WS-BPEL specification that allows specific steps in a business process to be assigned to and completed by an individual. Human tasks in business processes are wired to other activities directly. The runtime support for human tasks is provided by the human task manager component. With human tasks, escalation chains can be declared, work items can be transferred, suspended or resumed for robust support in an enterprise environment. Integration with user registries allows for tasks to be easily assigned to different groups of individuals.



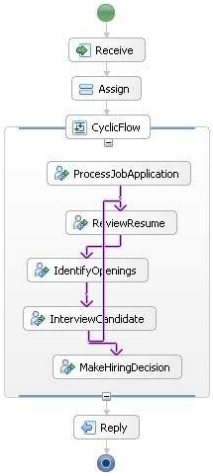
Snippets are an IBM extension to the WS-BPEL specification. Formerly known as a Java snippet, this has been extended to include support for other types of snippets as well. WebSphere Integration Developer includes a visual snippet editor, or you can program directly in Java. Snippets can be used to perform more robust processing on variables and business process state when the intended action cannot be performed using assign activities. An example is concatenating or parsing a string, such as changing the format of a person's name. Snippet activities should not be used for performing any type of external calls or processing.

Cyclic flows

Activity

Cyclic Flow

- **IBM extension to WS-BPEL**
- Provides graph-oriented modeling of business processes
  - Enables arbitrary cycles (back-links) in a flow
- Must contain exactly one start activity
  - A “start activity” is any activity without a link *to* it
- May contain any number of end activities (where at most one is actually reached)
  - An “end activity” is any activity without a link *from* it
- When there is more than one link from an activity, there must be transition conditions associated with the links



The cyclic flow activity is an IBM extension to the WS-BPEL specification. It is similar to the flow activity, except that it allows back-links. This enables the creation of arbitrary loops and cycles, which provides for graph-oriented modeling of business processes. A cyclic flow must contain exactly one start activity, where a “start activity” is defined as any activity without a link ***to*** it. Also, a cyclic flow can contain any number of end activities, where an “end activity” is defined as any activity without a link ***from*** it. In the human resources example on the slide, the start activity is “ProcessJobApplication” and the only end activity is “MakeHiringDecision”. When there is more than one link from an activity, there must be transition conditions associated with the links so that the business flow manager can determine which path to take next. Note that the order of evaluation of the transition conditions is arbitrary, so it is best to define mutually exclusive transition conditions when designing a cyclic flow.

Section

Summary

This section of the presentation provides a summary.

Summary

- WS-BPEL provides a standards-based language for defining the business process execution model independent of the implementation
- Business process choreographer (BPC) provides the runtime support for WS-BPEL
- IBM extensions provided for human task support, expressing business logic in Java, and quality of service enhancements for running processes in an enterprise environment

In summary, WS-BPEL provides a description language for defining business processes independent of the implementation. WebSphere Process Server and WebSphere Integration Developer provide support for almost all of the WS-BPEL 2.0 specification and address several issues that have been raised with it. WebSphere Process Server and WebSphere Integration Developer also provide several IBM extensions to WS-BPEL. People extensions allow you to specify human interactions. Java extensions allow for tighter integration with the underlying Java implementation. Quality of service extensions (for instance, microflows and long-running processes) support running business processes in an enterprise environment where transaction capabilities are an important consideration.

Feedback

Your feedback is valuable

You can help improve the quality of IBM Education Assistant content to better meet your needs by providing feedback.

- Did you find this module useful?
- Did it help you solve a problem or answer a question?
- Do you have suggestions for improvements?

Click to send e-mail feedback:

mailto:iea@us.ibm.com?subject=Feedback_about_WPSWIDv6_WSBPELDetails.ppt

This module is also available in PDF format at: ../WPSWIDv6_WSBPELDetails.pdf

You can help improve the quality of IBM Education Assistant content by providing feedback.

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 WebSphere

A current list of other IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml

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

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 or changes in the products or programs 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 (for example, 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.