TIBCO iProcess® Modeler

Integration Techniques

Version 11.9.0
August 2022

Contents

Overview

This section provides an overview of the possible integration methods offered by the iProcess Suite. The iProcess Suite can be implemented as a standalone system or it can form a component of an integrated business process solution.

Note: You can implement global transactional business processes if you use EAI steps and have the necessary resource managers or transaction processing monitors (TPM) such as BEA Tuxedo.

The iProcess Suite is an open product that can be used to integrate many systems. In most enterprises, there is always some form of integration required with third-party products such as:

- databases and data warehouses
- document management
- electronic mail
- eCommerce applications
- image processing

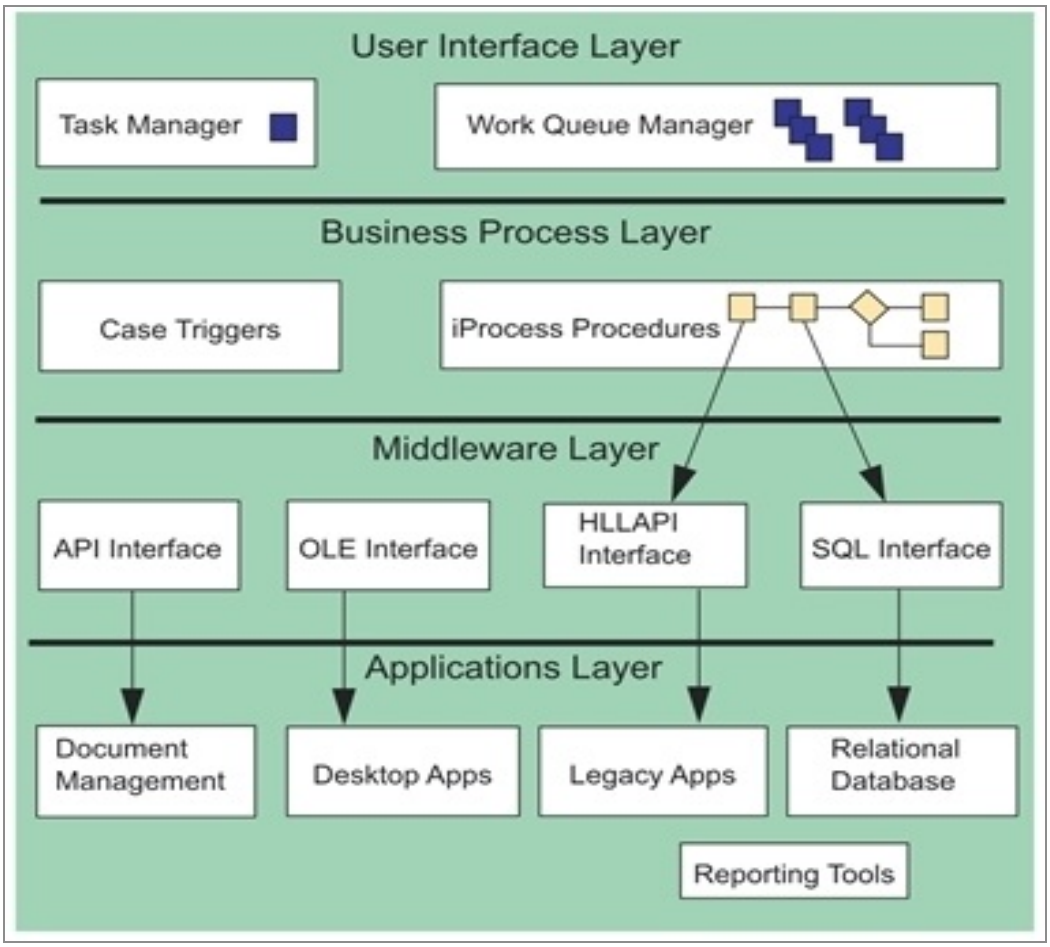
Enterprise Business Process Integration

The level of integration you need to perform depends upon the systems you use in your business processes and how you want these to be linked to your iProcess procedures. You can use any combination of the iProcess integration features for your solution.

Basic Integration Architecture

The basic model of any integration is shown in the following diagram. An integration can be split into the user interface, business process, middleware and application layers. iProcess performs the interfaces to the middleware components which in turn provide the interfaces to the specific applications.

Figure 1: Integration Layers For iProcess Applications



An integration can be split into the following layers:

- User Interface Layer (task manager, Work Queue Manager)
- Business Process Layer (case triggers, iProcess procedures)
- Middleware Layer (e.g. API, OLE, HLLAPI and SQL interfaces)
- Application Layer (e.g. document management, desktop applications, legacy applications, relation database and reporting tools)

iProcess performs the interfaces to the middleware components which in turn provide the interfaces to the specific applications.

User Interface Layer

The user interface has two main functions. It provides:

- a way of notifying users of new work items
- a task interface to guide them through completing the work items.

iProcess Workspace (Windows) uses the Work Queue Manager for the main interface for displaying work items. iProcess forms are used to provide the interface for the content of the work items.

You can provide alternative user interfaces using the Open Forms functionality. For example, work items are delivered to the Work Queue but when the user opens a work item a customized form is displayed. This could be designed in Visual Basic, Oracle Forms or Delphi, for example. See Using Open Forms.

Another method of providing an alternative interface is to deliver the work queue manager function by means of an application other than iProcess.

For example, you can implement the iProcess Application Layer (SAL) API or TIBCO iProcess™ Objects in external software. This means that work items are still delivered to iProcess work queues but the functions to manage those queues are controlled by bespoke applications. However, if you use this method, you also have to control the locking of work items, running of command scripts, and displaying a form populated with data retrieved from iProcess by calling SAL functions or TIBCO iProcess Objects methods.

Business Process Layer

This layer consists of one or more iProcess procedures that define the business processes. The TIBCO iProcess Engine processes the rules in the procedure and controls who does what and when, delivers appropriate work to the correct users and manages work items if it does not get processed in time.

There may be business process triggers or interfaces to other applications that can cause other instances of procedures to start such as when an event takes place (scanning a document or receipt of a file, etc).

Middleware Layer

The middleware layer is basically the interface to the external applications. The interface exposes pre-defined functions to iProcess and iProcess can call these by an agreed command line syntax. The interface is controlled by command arguments and usually exchanges data with iProcess using simple text files.

Application Layer

This layer contains all the external applications (such as relational databases, document management software, etc.) that contain information which needs to be delivered to users during a procedure, or updated by iProcess. For example, an Oracle database is used to store case data; the middleware layer provides the generic SQL interface to enable iProcess to retrieve and update information in the database.

Integration Levels

There are four basic levels of integration:

- Zero Integration
- Automated Application Delivery
- Fully Automated Procedures
- Embedded Business Process

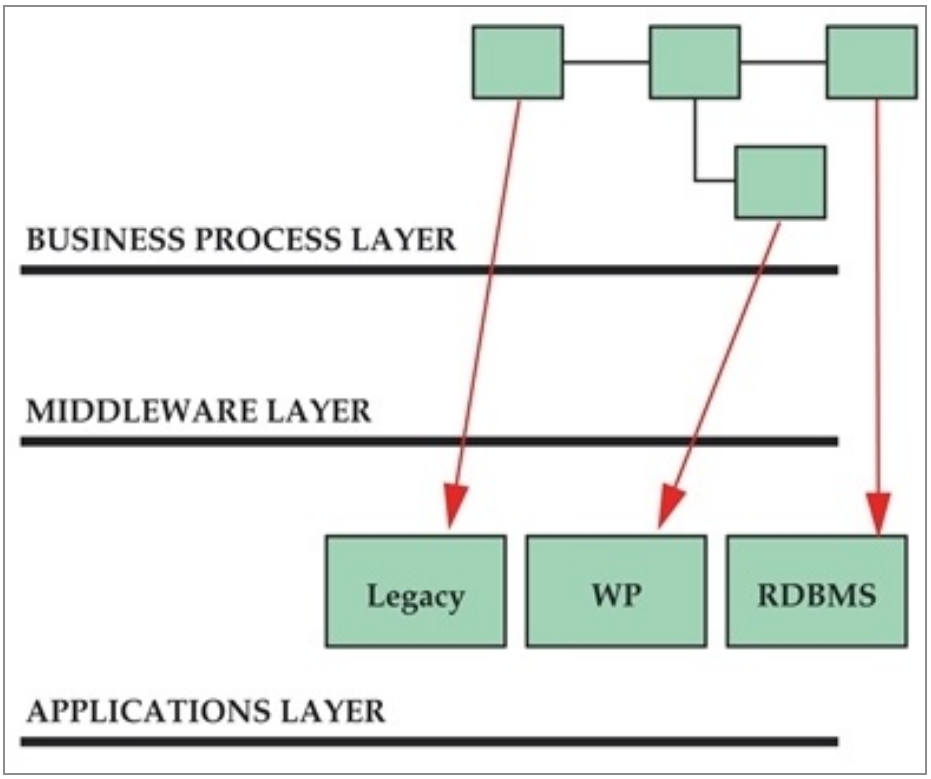
There is always the option of extending the level of integration once you have completed the initial phase of the project.

Zero Integration

This is where iProcess is used as a standalone business process solution. No integration is developed with other applications so the procedures just guide users through a business process using the standard iProcess forms.

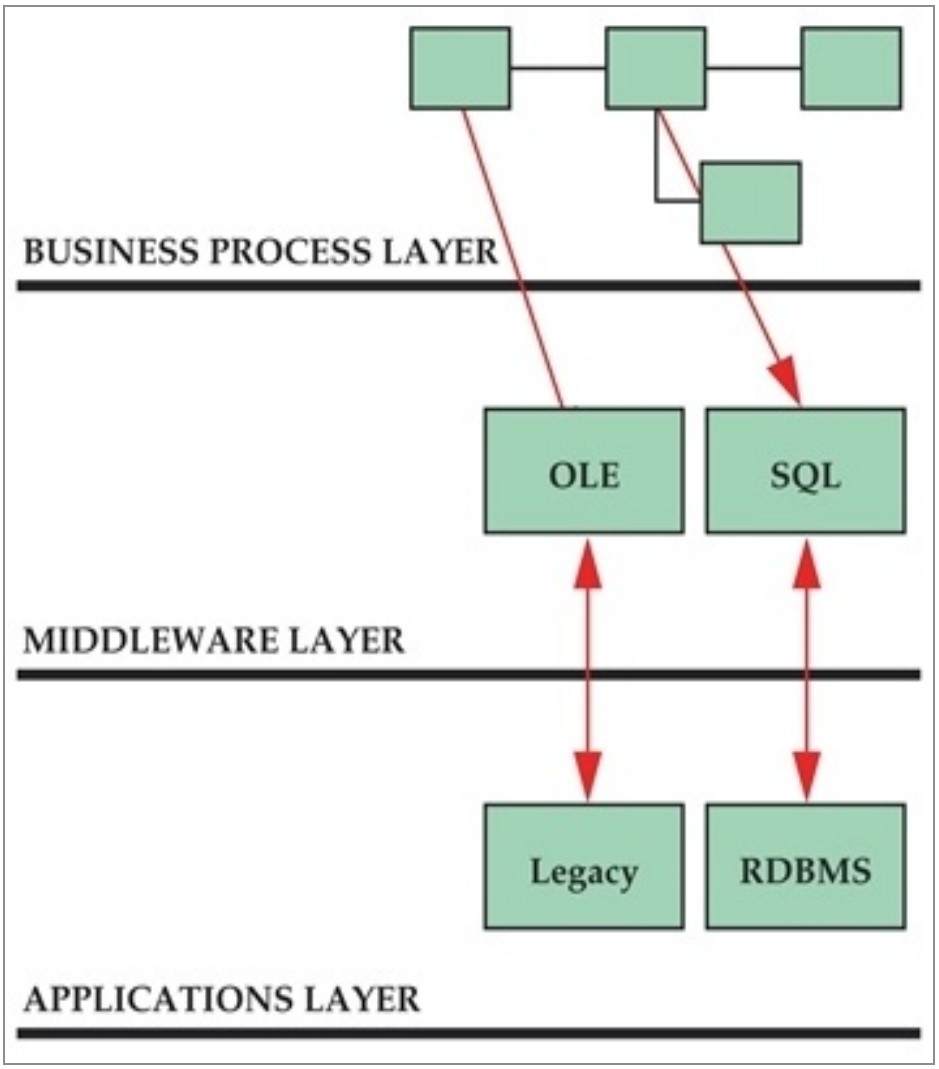
Automated Application Delivery

This is where iProcess invokes applications as required by the user. The user is guided through a business process and other applications are ‘popped-up’ automatically when required - for example, a word processor or spreadsheet.



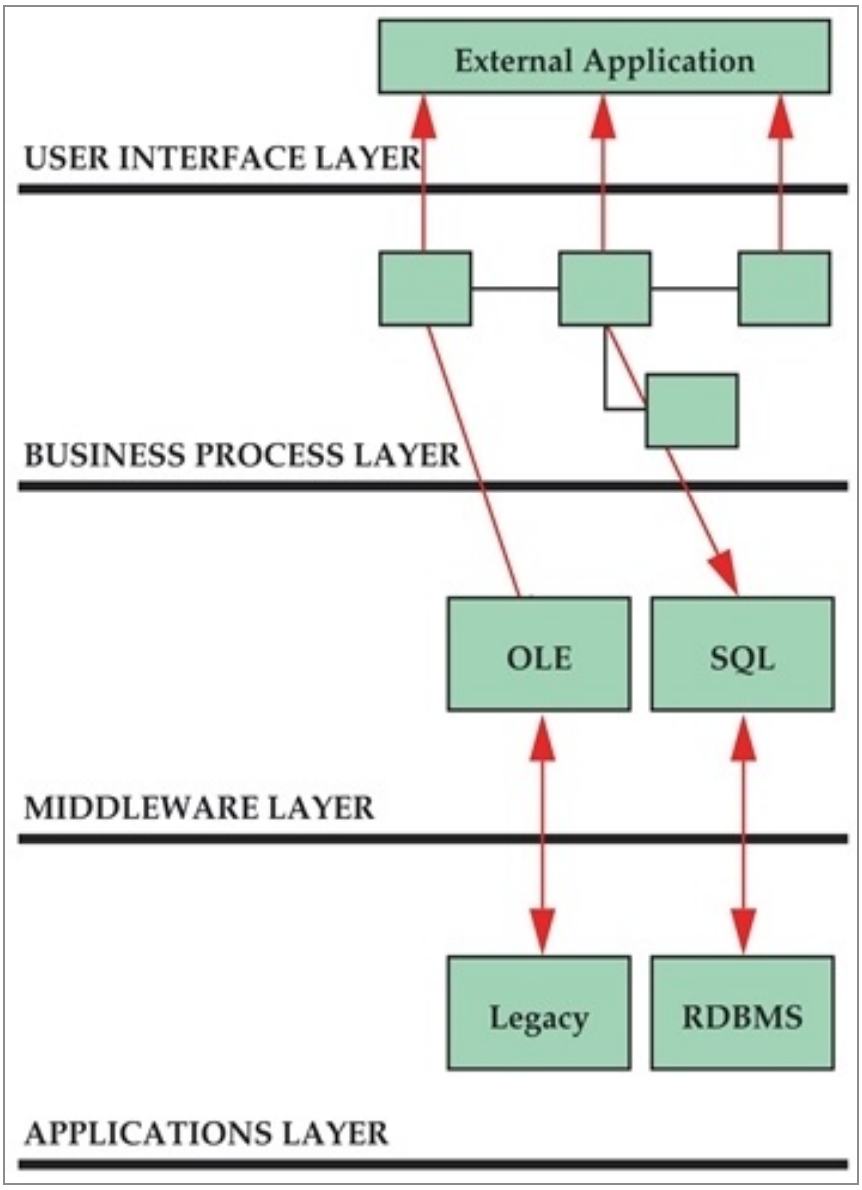
Fully Automated Procedures

This is where iProcess is fully integrated with other applications. The user is guided through a business process where tasks that involve other applications are automated, e.g. letter generation, database updates.



Embedded Business Process

This is where iProcess delivers work to external clients. The TIBCO iProcess Engine is used to control the business process but external applications are used to control the work to be done.



Integration Features Quick Reference

The following features are available for integrating iProcess procedures with the applications you use in your business processes:

- Enterprise Application Integration (EAI) Steps
- Graft Steps
- SSOLite Stored Procedures

- Open Forms
- Caseless Forms
- Events
- iProcess Activity Monitoring
- Open Work Queues
- Scripts
- iProcess Functions
- Dynamic Data Exchange (DDE)
- VBA
- XPDL
- The CUTIL Utility

Enterprise Application Integration (EAI) Steps

An EAI step enables iProcess to control updates to external applications and receive data back from applications to use in the procedure. If your system environment is set up so that all the disparate systems are in the same transaction environment, the external data updates and iProcess case data updates can be performed as part of a single global transaction.

The benefits of using EAI steps include:

- straight through processing of procedures with little or no user intervention. This is ideal for processes such as share trading and call center logging.
- transactional control of data updates. This ensures that all the data is updated to preserve data integrity, or no data is updated to leave the system in its previous state.
- controlling an external process or program, and to pass case data to it as a background task. The step can also get data back from the external application to use in iProcess.

For more information, see Using Enterprise Application Integration (EAI) Steps.

Graft Steps

Graft steps enable you to attach multiple sub-procedures or external processes to a specific point in your procedure. An external application is used to start a number of sub-processes using TIBCO iProcess Objects calls and these are grafted to the main procedure using a graft step. The graft step acts like a wait because the step is not released until all of the sub-processes have completed.

You would use Graft steps when you want to start an arbitrary number of sub-processes when the case is run depending on what case data is entered. A graft step can only work by using an external application and using TIBCO iProcess Objects calls.

For more information about setting up a Graft step, see Using Graft Steps. For more information about using TIBCO iProcess Objects calls to start graft steps, see the TIBCO iProcess Objects Client help.

SSOLite Stored Procedures

SSOLite is a set of stored procedures, available in the iProcess database, that provide applications with direct access to a limited subset of iProcess functionality.

An application can use SSOLite stored procedures to issue instructions directly to the iProcess background processes (by inserting messages into the iProcess message queues) to perform the following iProcess operations:

- start a case.
- trigger an event.
- graft a sub-procedure to a procedure (at run-time).
- jump a case to a different point in the procedure.
- suspend a case.
- re-activate a suspended case.

For more information, see the appropriate *TIBCO iProcess Engine (Database) Administrator's Guide*.

Open Forms

iProcess enables you to use a different forms application in place of the standard iProcess forms. For example, you may want to keep to a corporate user interface.

For more information, see Using Open Forms.

Caseless Forms

iProcess provides the ability to open a form for any step of a procedure without using a work queue item and without having to start a case of that procedure. This is useful when you need to open many forms that detail information from disparate sources during one step. For example, an insurance system may include a step that requires multiple information sources such as:

- claim history records
- risk assessment details
- insurance company vehicle information.

For more information, see Using Caseless Forms.

Events

Events can be used to respond to external events occurring in systems outside of iProcess. For example, a case can be suspended until an external event occurs. You can also use events to update case data using Process Variables - for example, updating CDQP parameters.

For more information, see Using Event Steps.

iProcess Activity Monitoring

The TIBCO iProcess Engine can be enabled to publish iProcess Engine activity information to external applications. An activity is any instruction in the iProcess Engine that creates an audit trail entry, for example, **Case started** or **Event Issued**. You can configure any combination of step and/or activity to be monitored. This enables an external application to monitor important business events during the processing of cases.

For more information, see Configuring Activity Monitoring.

Open Work Queues

iProcess enables work items to be opened from outside of iProcess using iProcess Workspace (Windows) **staffw.exe** options from the command line. This means that external applications can interact with work items, such as opening a specific work item, updating the work queue item list or setting a filter on the work queue items list.

For more information, see *TIBCO iProcess Workspace (Windows) Manager's Guide*.

Scripts

A script is a collection of statements that can be called from various places within a procedure. Scripts are most useful where a sequence of operations, perhaps external to iProcess, need to be invoked. An example may be passing data to an external credit checking system as part of a loan application.

For more information, see "Using Scripts" in the *TIBCO iProcess Modeler Advanced Design* guide.

iProcess Functions

iProcess provides many functions that can be used on their own or in scripts to perform specific operations such as running external programs or requesting data from external applications.

For details of all the functions and their parameters, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Dynamic Data Exchange (DDE)

iProcess enables you to use the DDE protocol to communicate with other DDE compatible programs. Data can be passed between two Windows applications while they are running.

For more information, see *TIBCO iProcess Expressions and Functions Reference Guide*.

XPDL

You can transform iProcess procedures into the Workflow Standard XML Process Definition Language (XPDL) and vice versa. This means that, as long as the vendor supports XPDL, process definitions can be used by different vendors, regardless of the workflow system that was used to produce them originally. For example, you can take a process definition that has been created by third party vendor, Enhydra™ JaWE (Java Workflow Editor) and load it into iProcess and vice versa.

For more information, see Transforming iProcess Procedures into the Workflow Standard XML Process Definition Language (XPDL).

The CUTIL Utility

The `CUTIL` utility, which is located in the *IPWW_HOME* directory, enables you to export procedures and hierarchical call tree of procedures from the client side. The benefits of using the `CUTIL` utility include:

- Export procedures or libraries to an XPDL file from the client side.
- Generate hierarchical reports for procedures, sub-procedures, sub-procedure parameter templates, and libraries. You can view the structure and relationships of the procedures, sub-procedures, templates, and libraries.

For more information, see Using the CUTIL Utility to Export Procedures and Call Hierarchies of Procedures.

Using Enterprise Application Integration (EAI) Steps

This section describes how to use EAI steps.

Overview

You can use EAI steps to interact with external applications so that iProcess procedures can integrate with other systems in your business process. For example, you can make updates to stock items in an external database or request an inventory via a web service. You could also integrate a document management system into the process and an existing legacy system.

EAI steps can send iProcess case data to external applications using whatever communication method is required. They can also pass data back from the application to iProcess.

If your applications are running in a transaction environment, you can set up the business process so that data updates are performed as a single transaction.

EAI steps are automatically processed by an iProcess background process (BG) instead of being sent to a iProcess user for completion. The EAI steps will be performed in-process, by the background, and can therefore participate in the business process transaction.

Therefore, EAI steps enable you to create automated business procedures that can be:

- closely integrated with third-party applications such as databases and legacy systems.
- designed to operate under transaction control.
- processed with little or no user intervention so that procedures are processed quickly (known as Straight Through Processing or STP).

Different types of EAI steps are available to communicate with different applications using their proprietary programming interface, for example:

- Web Services - enables you to call a web service from your procedure passing iProcess data to it and optionally receiving data back from it. For more information, see *TIBCO iProcess Web Services Plug-in User's Guide*.
- SQL - enables communication with SQL stored procedures. For information about installing the plug-in and creating the SQL EAI step, see *TIBCO iProcess SQL Plug-in User's Guide*.
- COM - enables communication with COM+ applications. For information about installing the plug-in and creating the COM EAI step, see *TIBCO iProcess COM Plug-in User's Guide*.
- EAIJAVA - enables you to design an iProcess procedure to call out a custom Java object. For more information, see *TIBCO iProcess® Java Plug-in User's Guide*.
- EAIJAVASCRIPT - enables the creation of server-side script steps that provide integration with external system or local server-side processing capabilities. For more information, see *TIBCO iProcess® JavaScript Plug-in User's Guide*.
- EAIBW - provides features to model and automate user-centric business processes. Also provides an easy to use interface between BusinessWorks and iProcess Engine. For more information, see *TIBCO iProcess® Connector for ActiveMatrix BusinessWorks™ User's Guide*.

The EAIDB steps are used to communicate with database stored procedures. TIBCO iProcess Engine only supports for connecting to database by using iProcess Engine database user (swuser) account, so you need to grant execute permission to swuser on all custom stored procedures used by EAI Database steps.

How an EAI Step Works?

Procedure definers use an EAI step as one (or more) of the steps in their procedure. The EAI step defines what external system is being updated and what data to update. Different types of EAI step are used depending on the external system being used. For example, iProcess provides a Web Service and a COM EAI step type.

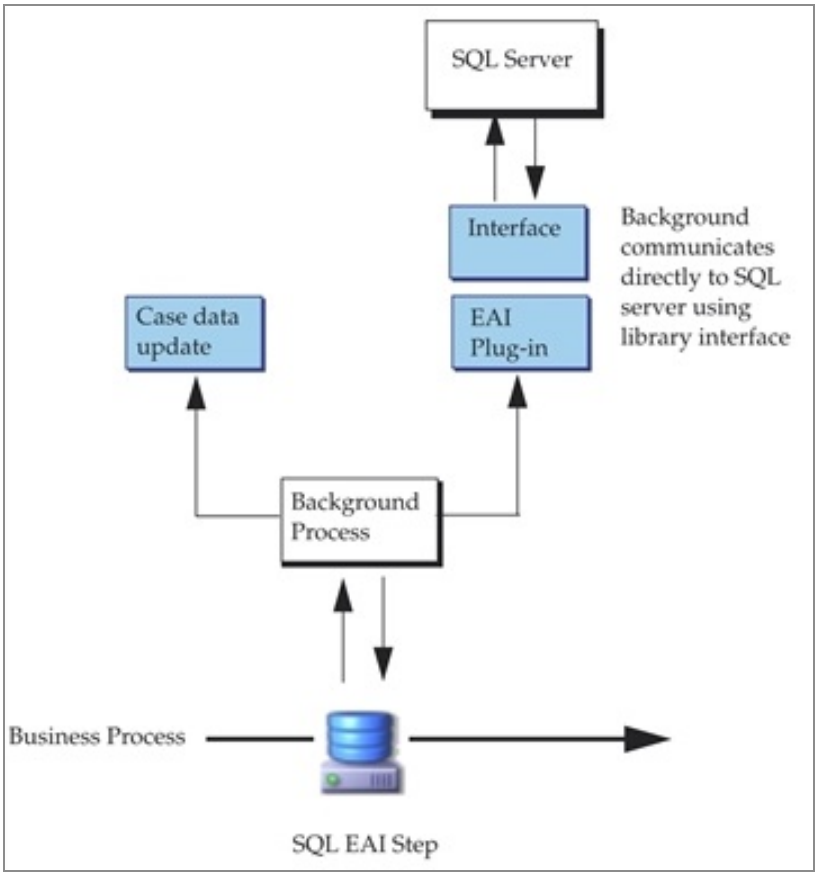
Each EAI step has a corresponding run-time (server) plug-in library on the iProcess Engine that controls the interface calls to the
external application. For example, the SQL EAI step contains the necessary interface calls to communicate with stored procedures on the SQL server.

To function properly, the plug-ins need to be registered on each server that is processing a procedure containing EAI steps. For most EAI plug-ins, this is performed when the TIBCO iProcess Engine is installed. Alternatively you can use the *SWDIR***util****sweaireg** utility. You can, however, design procedures using an EAI client plug-in for which you have not installed the corresponding EAI server plug-in. For more information, see "Managing EAI Step Server Plug-ins" in *TIBCO iProcess Engine Administrator's Guide*.

When a case of the procedure is started:

- the EAI plug-in extracts the case data from the step and sends it to the external application.
- the EAI step can also extract data from the external application to use in the case and can be set up with delayed release so that it is not released immediately.

The following diagram shows an example of how an EAI step communicates with an external application.



How EAI Steps Affect the Processing of a Case?

When using EAI steps in your procedure, you need to be aware that case processing is affected by the type of EAI processing used by the EAI steps. When the case gets to an EAI step, an iProcess background process makes an EAI call-out to an external system; that background process cannot process other cases (or other branches of the procedure) while waiting for an EAI call-out to complete.

Note: Other background processes running on the iProcess Engine can still process other work.

The EAI call-out is the communication of a request to an external system and the communication of the response to that request by the external system. In general, the procedure can be designed to utilize EAI steps in one of two ways:

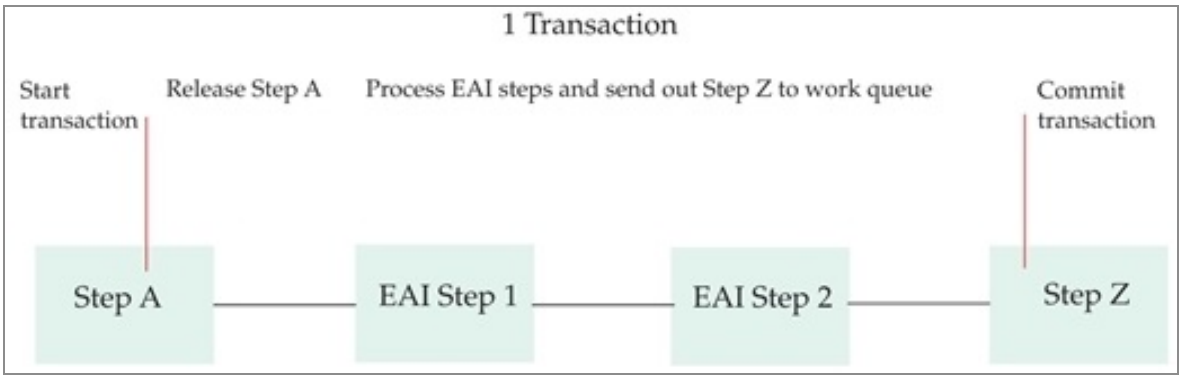
- Immediate Release (default method)
- Delayed Release (only if supported by the plug-in and this step is defined to use delayed release).

The EAI step definition in the procedure defines whether the step is immediate or delayed release.

Immediate Release

When the EAI call-out completes (i.e. a request is made and the reply is received), the BG process reads the EAI step definition to determine if the step should be released immediately.

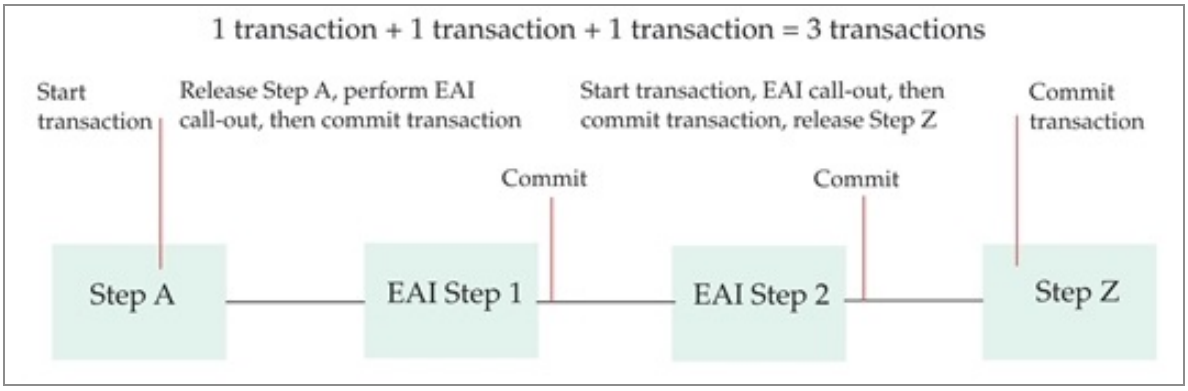
If the EAI step definition is not set to delayed release, the step's actions are processed immediately as part of the same transaction. Deadlines are not supported when using Immediate Release.



You can define a time threshold for the duration of processing an immediate release EAI step. If the time for processing an EAI step is over the defined threshold, the EAI step keeps processing, and the time period of processing the EAI step is logged in the `sw_warn` file. For more information, see "EAI_STEP_TIMEOUT" in *TIBCO iProcess Engine Administrator's Guide*.

Delayed Release

If the EAI step definition is defined to be delayed release, the actions of the EAI step are not processed immediately when the EAI call-out completes. They are processed when a release instruction is given to the BG process at a later time and in a separate transaction.



Delayed release should be used where the request on the external system can take a long time to complete. In general, delayed release call-outs will queue a request for some work to be done by an external system. The external system picks up this request, processes it and then uses TIBCO iProcess Objects or **pstaffifc APPRELEASE** to tell iProcess that the step is complete. If you need to keep the external system in synchronization with the iProcess case then the EAI plug-in and external system must be able to support withdraw requests and remove the appropriate externally queued request.

> **Note:** If an incorrect or terminated case reference is entered into the **APPRELEASE** command to manually release a step, **APPRELEASE** does not return an error message indicating that it could not process the command.

If you have an EAI call-out that takes 5 minutes to return some query results, you do not want to block the background process for 5 minutes while it waits for the response. In these situations, you need to use delayed release so that the BG process can continue processing other work.

Some EAI plug-ins may not support delayed release (i.e. the application does not enable the procedure definer to select it). In this case, the Delayed Release tab should be disabled (grayed out) in the EAI Properties pane. Some EAI plug-ins may force all call-outs to be delayed release if the external system always requires this.

EAI Call-Out Methods

The following two methods can be used to invoke EAI call-outs:

- Synchronous
- Asynchronous With Reply

The method used to invoke the EAI step depends on how the EAI plug-in has been designed and is internal to the EAI plug-in.

Synchronous

When EAI call-outs are made using synchronous invocation, the iProcess background process is blocked until the EAI call-out has completed. Only one synchronous EAI call-out can be invoked at one time by each BG process i.e. even if you designed a procedure with EAI steps in parallel (see Creating Procedures with Parallel EAI Steps), each EAI step is processed sequentially in turn. Compare this with asynchronous with reply invocation where you can make simultaneous EAI call-outs when they are in parallel.

Asynchronous With Reply

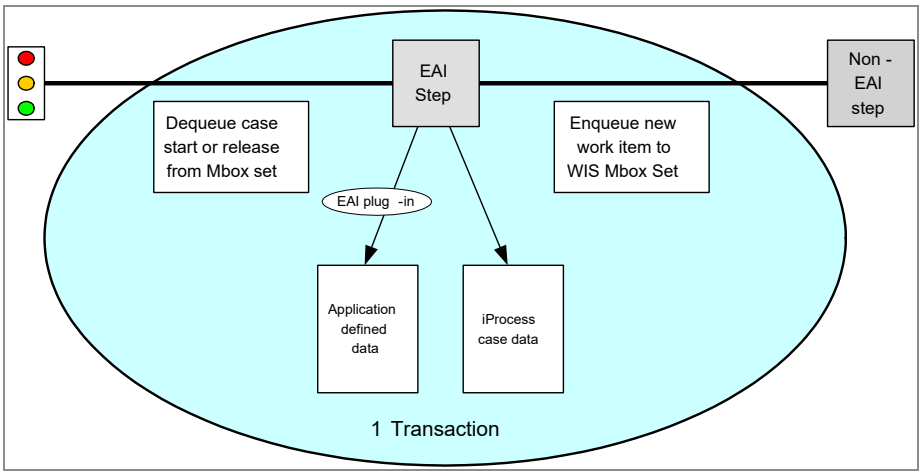
This call-out style de-couples the request and response into two separate operations from the iProcess background process: the initiation of the call-out and the “get reply” of the call-out. EAI call-outs can be made in parallel (by designing the procedure with parallel EAI steps - see Creating Procedures with Parallel EAI Steps) and they all are invoked. The call-out is initiated for all parallel EAI steps before the BG waits for a reply from the first to complete. If the replying step is immediate release, its actions are processed before waiting for replies to any other asynchronous with reply call-outs.

Using EAI Steps in a Transactional Business Process Environment

In many enterprises, business processes are controlled by many disparate systems including databases, document management systems and legacy systems. It can be difficult to integrate the systems to keep control of a business process and make sure that all of the data is kept synchronized. Data updates to various parts of the system can fail due to systems being down while other updates have been successful. This can lead to integrity errors in the business process such as bank accounts being debited but not credited.

EAI steps can be used to make updates to third party systems under transaction control. For example, a COM+ EAI step can call out to a COM application that performs some data checks. The MSDTC transaction manager program can register the COM application as being part of the same transaction as iProcess so that the business process can operate as a single transaction.

Each EAI step can only be part of the same transaction that is controlled by the iProcess background process. By default, the iProcess background process bundles all consecutive/concurrent EAI steps into the same transaction. If there is a failure in one step then all steps are rolled back. However, you can control the granularity of the transaction for the EAI steps using one or more Transaction Control (TC) steps. You can break up a single transaction into multiple transactions as you require or design the procedure flow so that one branch can start in one transaction while another branch is started as another transaction. For more information about controlling transactions using a TC step, see Using Enterprise Application Integration (EAI) Steps.



By combining multiple EAI steps in sequence or in parallel, multiple external updates can be processed quickly by the background case instruction processes. The external resources can be completely separate systems involving a mixture of new and legacy systems. If these are operating in the same transaction environment, iProcess can control the entire business process in a single transaction.

Designing Procedures With EAI Steps

You can alter the way that EAI steps work in your procedure by the way you place them on your procedure definition. You can perform the following activities:

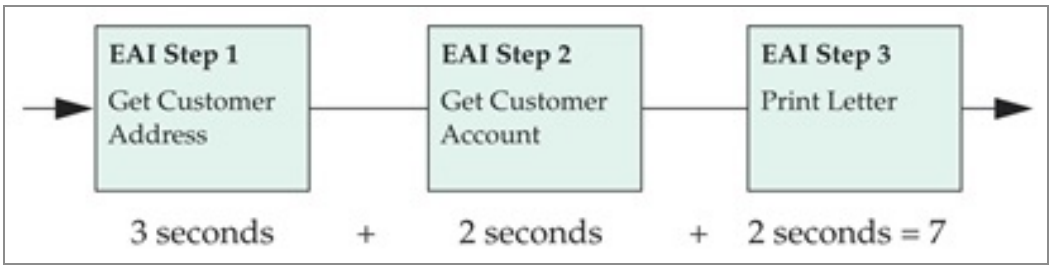
- place EAI steps in parallel so that the steps can be invoked simultaneously. See Creating Procedures with Parallel EAI Steps.
- create straight through procedures (STP) by linking EAI steps in sequence. See Creating Straight Through Processing (STP) Procedures.
- define a delayed release setting so that the external application executes the EAI step at a later time. See Delaying the Release of EAI Steps.
- use the SW_QRETRYCOUNT system field to provide exception handling for transactions with failing EAI steps. See Using the SW_QRETRYCOUNT System Field to Provide Exception Handling for Transactions With Failing EAI Steps.

Creating Procedures with Parallel EAI Steps

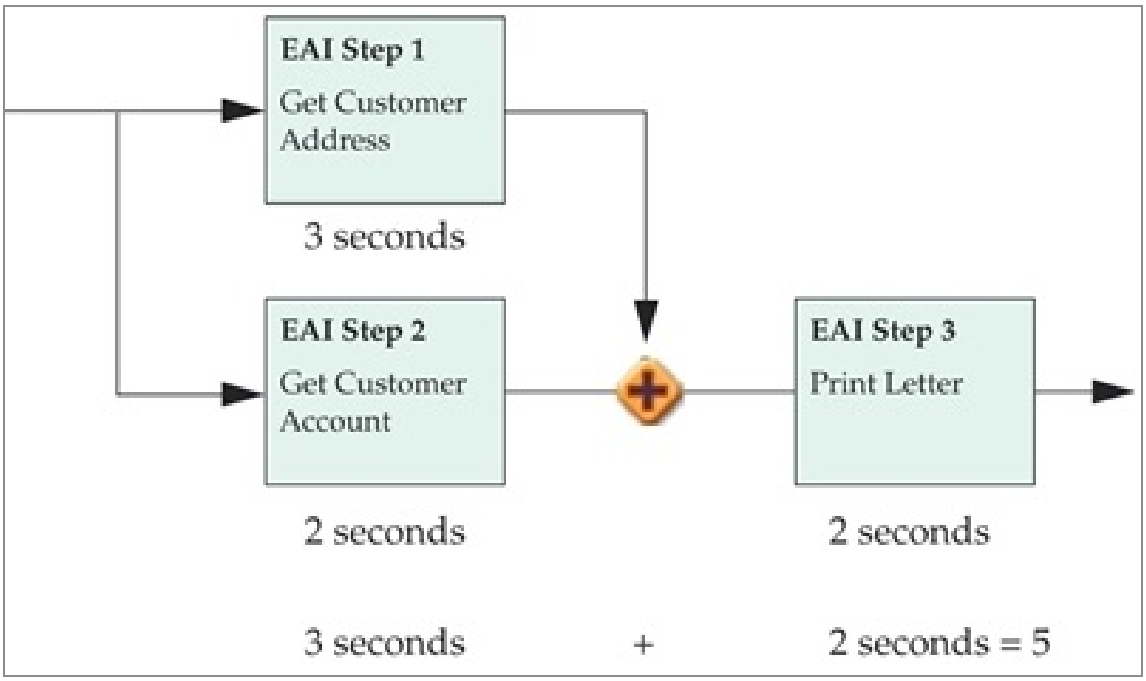
EAI plug-ins that support the Asynchronous With Reply call-out method can be utilized to increase the performance of processing complex procedures. Asynchronous with reply means that the call-out and reply is split into two operations. This means that multiple EAI call-outs can be made in parallel before asking for the replies.

When a reply to one of the asynchronous with reply call-outs is received for a step defined as immediate release, the actions of the EAI step are processed immediately. This is performed before waiting for replies to any other asynchronous with reply call-outs that are actions of the releasing EAI step.

The background process checks for other EAI steps on parallel workflow paths and call-out to these in parallel. The following procedure shows a series of steps in sequence that takes 7 seconds to complete (3 seconds to process step 1, followed by 2 seconds for step 2 and 2 seconds for step 3).



You can decrease the time this case takes by putting the EAI steps in parallel so that they are invoked at the same time. The following diagram shows how the procedure can be redesigned so that two EAI steps are processed in parallel and the procedure now takes only 5 seconds to complete (3 seconds for step 1 and 2 seconds for step 3; step 2 is simultaneously processed with step 1.)



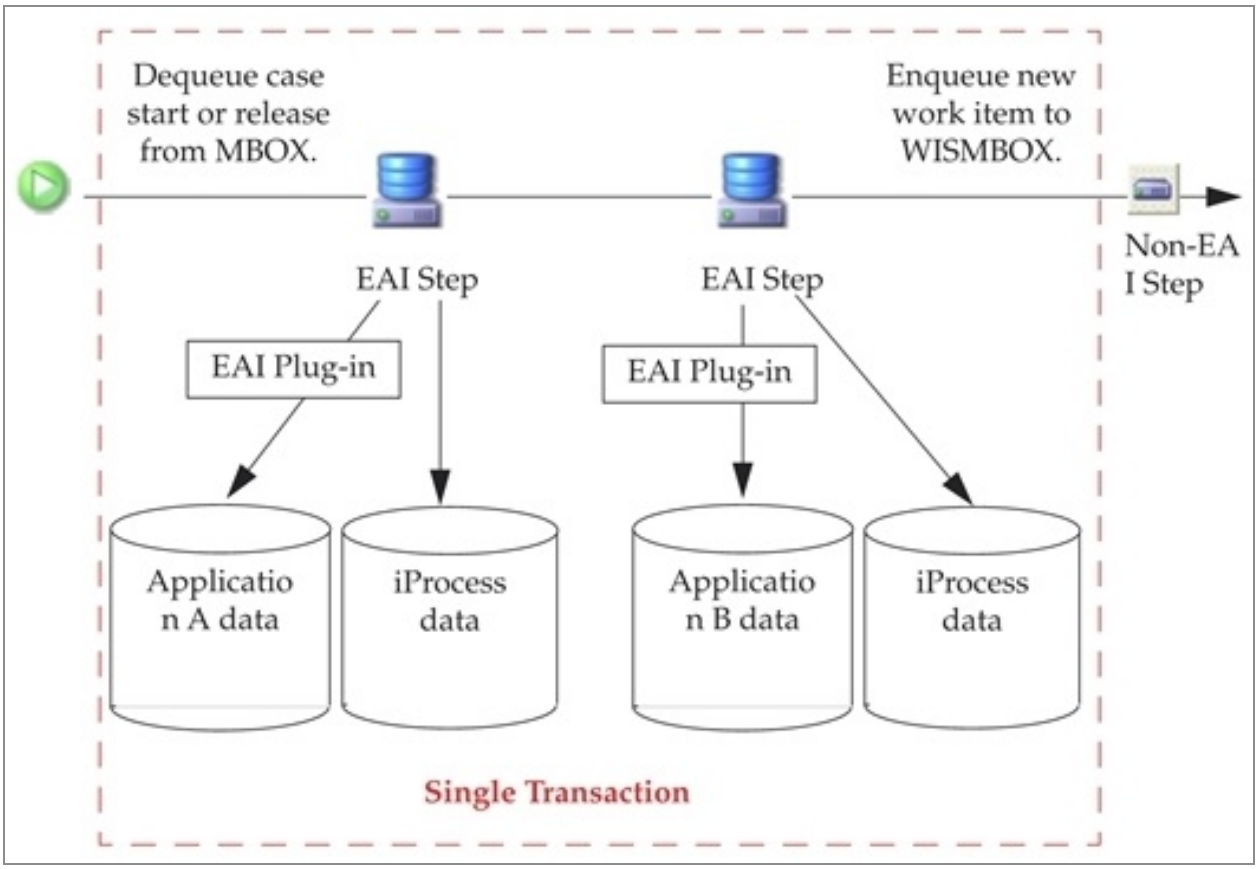
For example, suppose a procedure contains three EAI steps that are processed in sequence. Each take about 2 seconds to complete so a case of this procedure takes 6 seconds to complete. The procedure can be redesigned so that two EAI steps are processed in parallel and the case now takes only 4 seconds to complete.

Note: If the EAI plug-in does not support asynchronous with reply, the background process will run step 1 followed by step 2 followed by step 3 as in the first procedure (synchronous behavior - see Synchronous).

Creating Straight Through Processing (STP) Procedures

You can set up a Straight Through Processing (STP) procedure by linking sequential EAI steps together. This type of procedure can be processed by iProcess with little or no user interaction and the entire procedure can be performed as one transaction. This type of procedure design can enable business processes to be completed very quickly so it can be useful for processes such as share trading, customer call center queries and financial transactions.

Because multiple iProcess background processes can be running, EAI steps can be used to directly call out to third party applications. Multiple EAI steps can be processed sequentially making processes faster to complete.



Delaying the Release of EAI Steps

You can delay the release of an EAI step so that the external application releases the step at a later time or when it has completed its processing. For example, if an external database is awaiting some customer information, the step is not released until the database is updated. If this update takes a while, you can prevent the background process being blocked during this time by using delayed release.

Delayed release EAI steps do not immediately process their actions because iProcess saves the step as outstanding in the case information. The release can be triggered by the external application via TIBCO iProcess Objects calls or using the **Pstaffifc APPRELEASE** command.

Note: If an incorrect or terminated case reference is entered into the **APPRELEASE** command to manually release a step, **APPRELEASE** does not return an error message indicating that it could not process the command.

Using the SW_QRETRYCOUNT System Field to Provide Exception Handling for Transactions With Failing EAI Steps

The TIBCO iProcess Modeler's message queuing system ensures reliable delivery of iProcess messages, with transactional control and integrity provided by the underlying database resource manager.

Note: For more information about the messaging system, see *TIBCO iProcess Engine Architecture Guide*.

When, for example, a new case is started or a work item released, a message is enqueued to a message queue, containing the necessary instructions to process the transaction that forms the next step in the business procedure being executed. The message is dequeued from the message queue and processed by a **BG** process, and only deleted from the message queue when the transaction has successfully completed. If the transaction fails and is rolled back, the message remains in the queue, to be retried later.

Configurable parameters define how many times the message will be retried, and the delay between each attempt. If the retry limit is exceeded, the message is moved to the exception queue (also known as the dead queue or poison queue), and manual intervention by a system administrator will be necessary to resolve the problem and progress the case that the message belongs to.

Note: On Oracle variants, these parameters are provided by Oracle AQ parameters. On SQL and DB2 variants, they are provided by the **IQL_RETRY_COUNT** and **IQL_RETRY_DELAY** process attributes. The default values are a retry limit of 12 with a 300 second delay between retries.

Potential Messaging Problems When EAI Steps Are Used

When the data associated with a message is internal to the TIBCO iProcess Modeler, transactions generally succeed. However, if a transaction involves one or more EAI steps, it is possible for the transaction to fail repeatedly - either because the external system that the EAI step needs to communicate with is unavailable, or because the interface between the TIBCO iProcess Modeler and the external system has not been properly defined.

In pre-10.5 versions of the TIBCO iProcess Modeler, there is no visibility of a message's failure count. This means that:

- There is no way for an application to tell how many times a transaction involving an EAI step has failed, or indeed if the retry limit has been exceeded and the message moved to the exception queue.
- Exception handling logic cannot be built into the procedure that contains the EAI step to handle this scenario.

The SW_QRETRYCOUNT System Field

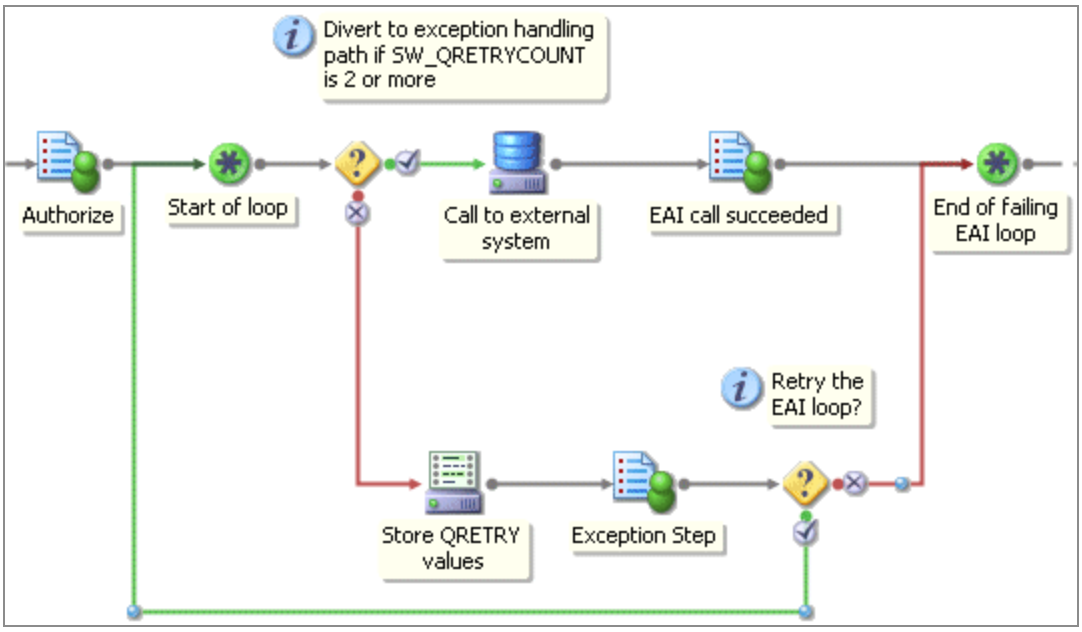
The **SW_QRETRYCOUNT** system field (introduced in Version 10.5) returns the number of times that a message in a message queue has failed. The field's value is **0** the first time a message is processed, and is incremented each time the message fails. For example, if a **BG** process is processing a message and **SW_QRETRYCOUNT = 2**, this means that the **BG** is attempting to process the message for the third time.

Note: The **SW_QRETRYCOUNT** field only returns a meaningful value when it is used during the processing of a message by a **BG** process. If it used in any other circumstance (for example, displayed on a form) it will return **SW_NA**. If you want to display the value in a form or use it elsewhere in the procedure you must first use an EAI Script step to assign it to another field, as part of the same transaction.

Using SW_QRETRYCOUNT to Provide Exception Handling for EAI Steps

The procedure extract below shows a simple example of how you can use **SW_QRETRYCOUNT** to build suitable exception handling into a process. Following the release of the **Authorize** step, an EAI step (**Call to external system**) is used to perform an update to an external system. To allow for the possible failure of this step, a condition has been inserted immediately before it which checks the **SW_QRETRYCOUNT** value for the

message. Once this value reaches **2**, the procedure diverts down an exception handling path.



When the **Authorize** step is released, a message is enqueued in which the condition and **Call to external system** step are processed and, if the **Call to external system** step succeeds, the step following it is sent out.

When a **BG** process first processes this message **SW_QRETRYCOUNT** has a value of **0** so the **Call to external system** EAI step is sent out. However, if the **Call to external system** step fails the transaction is rolled back, **SW_QRETRYCOUNT** is incremented to **1** and the message is retried.

If the message fails again - for example, because the external system being called is still not available - **SW_QRETRYCOUNT** is incremented to **2** and the next retry will divert the process down the exception handling path. When this happens:

1. The **Store QRETRY values** (EAI Script) step assigns the current **SW_QRETRYCOUNT** value (in this case, **2**) to a numeric field that can be displayed in the **Exception Step** form. For example:

   ```
   SW_QRETRYCOUNT:=EAI_FAIL_COUNT
   ```

   This is necessary if you want to make the **SW_QRETRYCOUNT** value available on the **Exception Step** form, *because* **SW_QRETRYCOUNT** has no meaningful value outside the scope of the current message being processed by the **BG** process. If you simply mark **SW_QRETRYCOUNT** on the **Exception Step** form it will display as **SW_NA** when the work item is opened.

2. The **Exception Step** is sent to the system administrator. This completes the transaction and the message is deleted from the message queue. The **Exception Step** can, if required, use the **EAI_FAIL_COUNT** field to display the number of times that the EAI step has failed. It should also provide:
   - whatever other information is deemed necessary to assist the administrator in resolving the problem.
   - a **Yes**/**No** field or similar decision point to allow the administrator to specify whether or not the procedure should loop back to retry the **Call to external system** step again - depending on whether or not they are able to resolve the problem with the external system.

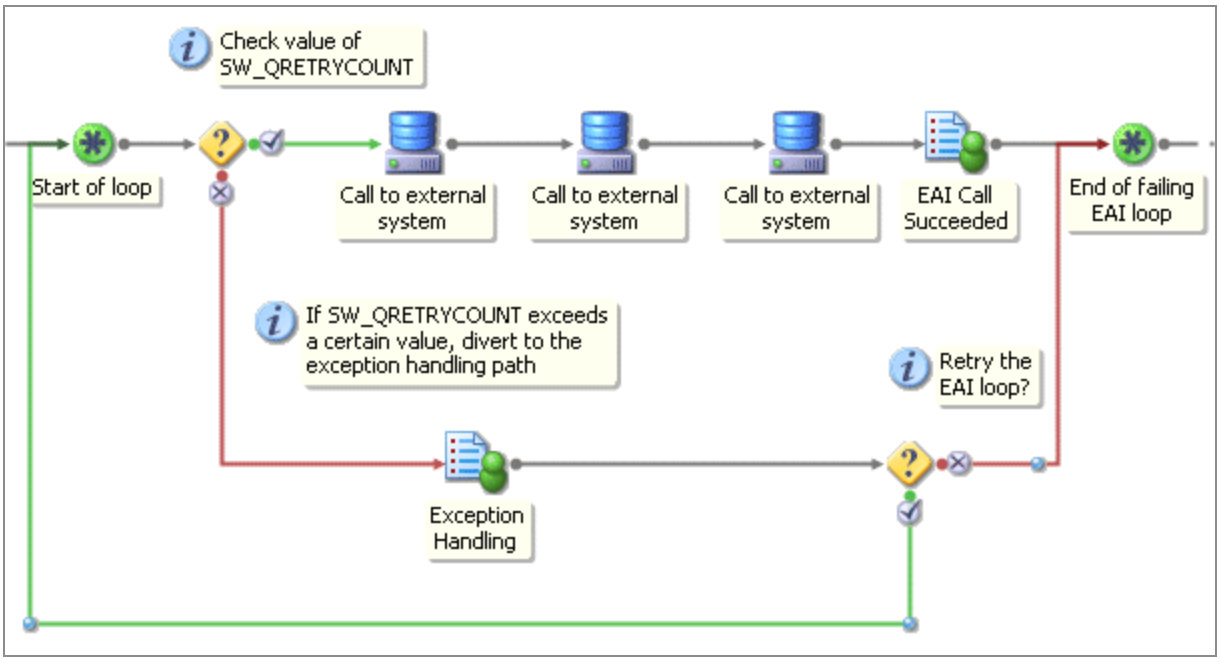
   Assuming that the administrator has investigated and successfully resolved the problem with the external system, they then release the **Exception Step**. A new message is enqueued and processed by a **BG** process:
3. The **Retry the EAI loop** condition is evaluated and sends the procedure back to retry the **Call to external system** step again.
4. This time, **SW_QRETRYCOUNT** has a value of **0**, because it is a different message. The **Call to external system** step is therefore sent out again and should now succeed, allowing the transaction to complete successfully and the case to proceed to the **EAI call succeeded** step.

There are a number of further things you should consider when using **SW_QRETRYCOUNT**, which the following sections discuss.

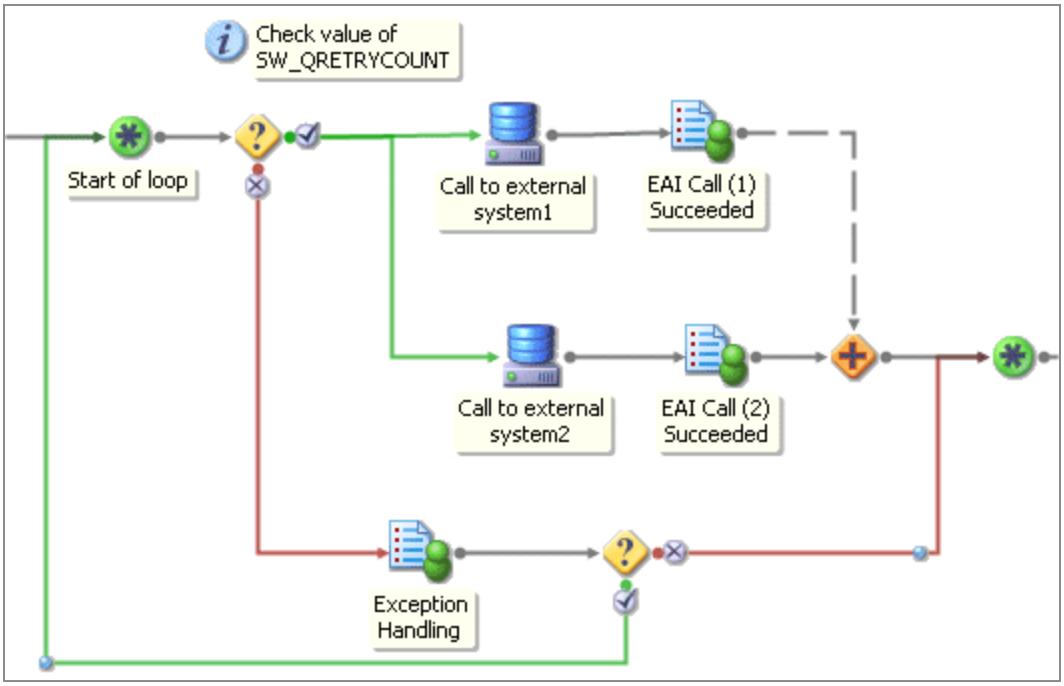
Handling Sequenced and Parallel Execution of EAI Steps

A continuous sequence of EAI steps, or a number of EAI steps executed in parallel, are all processed as part of a single transaction (or message) and will therefore have a single **SW_QRETRYCOUNT** value. Exception handling cannot therefore be implemented on a per-EAI step basis in this case.

In the example shown below, which uses a series of **Call to external system** EAI steps, the transaction will be rolled back if a single EAI step fails and **SW_QRETRYCOUNT** incremented for the whole series.



Similarly, in the following example two parallel **Call to external system** EAI steps are executed as part of the same transaction. If either of the EAI steps fail the transaction will be rolled back and **SW_QRETRYCOUNT** incremented. Neither of the **EAI Call Succeeded** steps will therefore be sent out unless both EAI steps succeed.



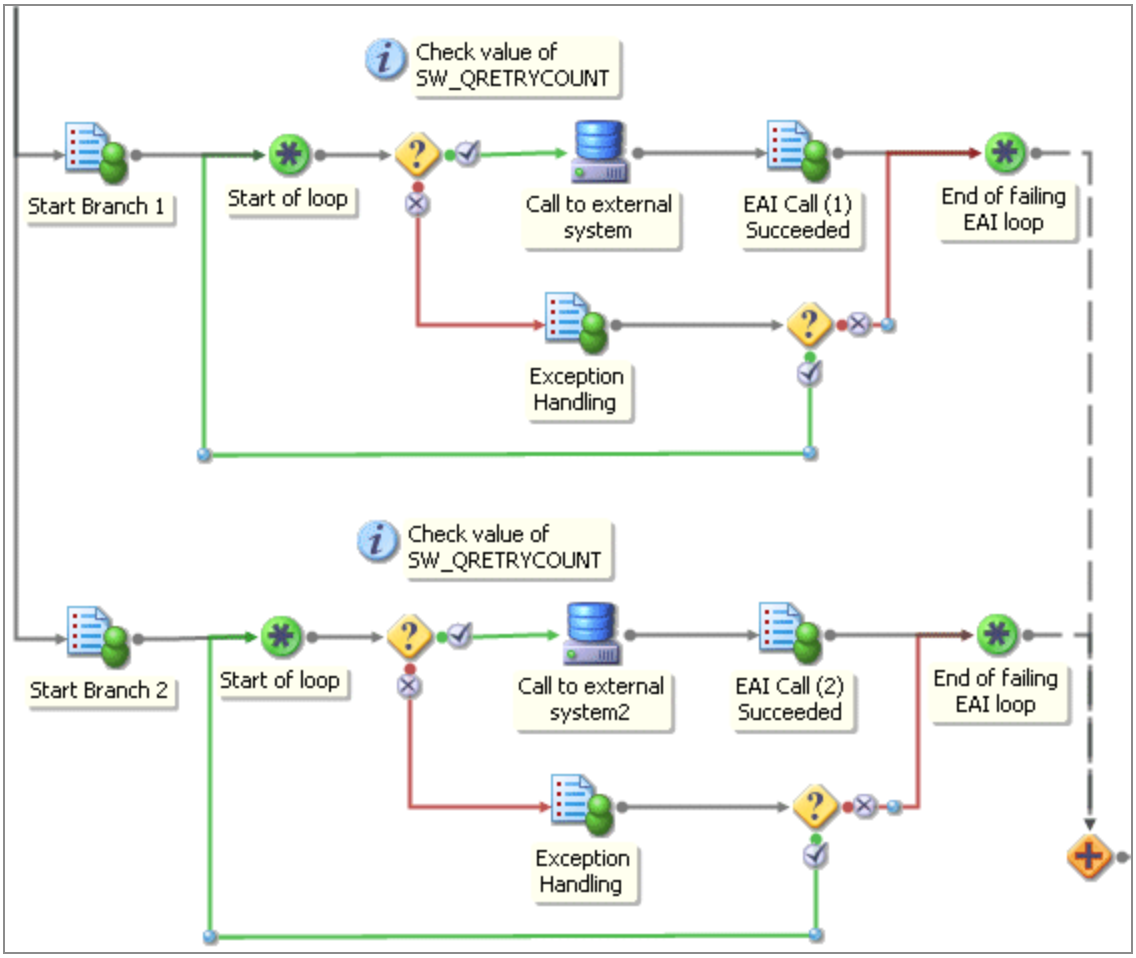
If you want to use **SW_QRETRYCOUNT** to implement exception handling on a per-EAI step basis, so that failing EAI steps are independent of each other, you must split the EAI steps

up so that they are processed as separate messages or transactions. You can do this by using multiple branches, preceding each EAI step either with a normal step (see Using Multiple Branches with Normal Steps), or with a Transaction Control Step (see Using Multiple Branches with Transaction Control Steps).

Using Multiple Branches with Normal Steps

In the following example, two parallel **Call to external system** EAI steps are executed in separate branches. Each branch is started by a different message, when the **Start Branch 1** and **Start Branch 2** normal steps are released. This means that:

- each message has its own **SW_QRETRYCOUNT** value, and exception handling can be implemented for each branch.
- the **Call to external system** EAI steps succeed or fail independently of each other. One branch can progress to the wait step even if the other is failing.

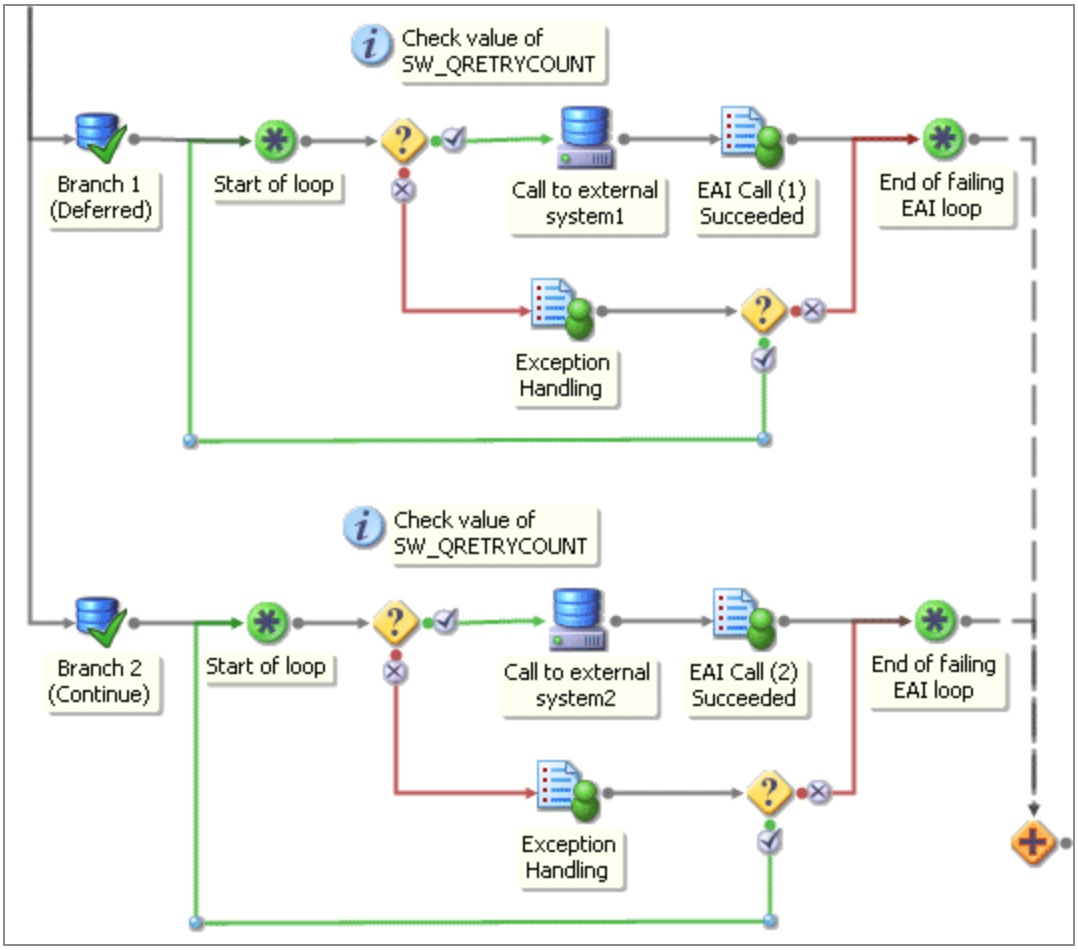


Using Multiple Branches with Transaction Control Steps

Transaction Control Steps can be defined to handle a commit operation in two different ways:

- *Commit and Concede*, where the **BG** process commits the transaction and posts a RELEASE message back to the queue.
- *Commit and Continue,* where the **BG** process commits the transaction but then immediately continues processing the rest of the message as a new transaction.

In the following example, two parallel **Call to external system** EAI steps are again executed in separate branches, but this time Transaction Control Steps are used to control the processing of each branch. Further, each branch uses a different type of Transaction Control Step to illustrate the difference in the way that each type is processed:



Each branch is processed as follows:

- *Commit and Concede (Deferred)* operation: When the **Branch 1 (Deferred)** step is released, the **BG** process commits the current transaction and posts a RELEASE

message back to the queue. Branch 1 is processed when the RELEASE message is dequeued and processed, and therefore has its own **SW_QRETRYCOUNT** value associated with the RELEASE message.

Note: The only difference from the previous example is that the message is generated from a RELEASE message generated by the background process, rather than by a foreground process releasing a work item.

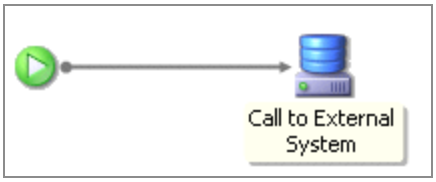
- *Commit and Continue* operation: When the **Branch 2 (Continue)** step is released, the **BG** process commits the current transaction, but then carries on and attempts to process Branch 2 - effectively as part of a new transaction. If the **Call to External System2** EAI step fails, the BG process rolls back to the **Branch 2 (Continue)** step, but cannot increment the **SW_QRETRYCOUNT** value because the message associated with that step no longer exists - having been removed from the message queue when the transaction was committed.

  However, the retry delay defined for the Transaction Control Step effectively sets a deadline on the step. When this deadline expires, a process case (PROCCASE) message is posted to the message queue. When this message is processed, Branch 2 is retried. If the **Call to External System2** EAI step fails again, the PROCCASE message still exists, so the **BG** process can increment the **SW_QRETRYCOUNT** value and processing of the branch can continue.

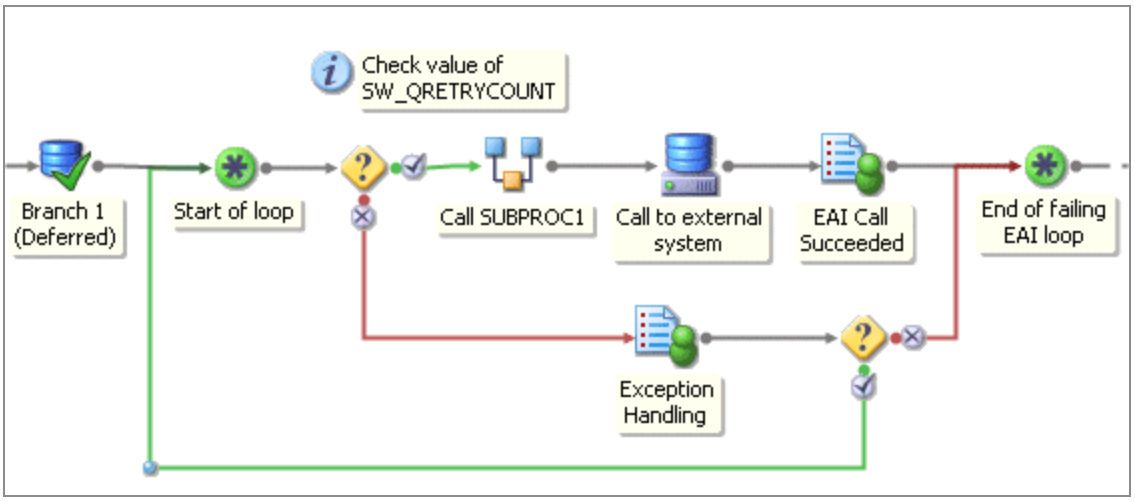
Note: In this case, **Branch 2** has effectively failed once already before the PROCCASE message is processed.

Using Sub-Procedures

Sub-procedures are processed entirely as part of their parent case transaction. For example, suppose that a **SUBPROC1** sub-procedure is defined to execute a single **Call to External System** EAI step, as follows.



Now assume that **SUBPROC1** is called from a procedure as shown below:



When the **Branch 1 (Deferred)** step is released, the **Call SUBPROC1** sub-procedure and **Call to External System** EAI step are both processed as part of the RELEASE message. If either the sub-procedure or the EAI step fails, the transaction is rolled back, **SW_QRETRYCOUNT** is incremented and the loop retried in the normal way.

Note:
If you are upgrading from or importing procedures from pre-10.5 versions of the TIBCO iProcess Modeler, you should note that the transaction boundary has changed in this case. This is because sub-procedures are now processed entirely as part of their parent case transaction, whereas in earlier versions the transaction would be committed when the sub-procedure completed successfully.

On a pre-10.5 version of the TIBCO iProcess Modeler, if **Call SUBPROC1** succeeded, the transaction would be committed and a sub-procedure done (SUBDONE) message would be posted to the queue. The following **Call to External System** EAI step would then be processed when the SUBDONE message was dequeued. It would therefore have its own **SW_QRETRYCOUNT** value associated with that message.

Defining an EAI Step

To define an EAI step in your procedure:

1. In the **TIBCO iProcess Modeler**, click the EAI Step tool then place it on your procedure definition.

2. The Properties pane is displayed.
3. Enter the **Name** and **Description** for the step.
4. In the **EAI Step Type** drop-down list, select the appropriate entry for the plug-in you want to use. (For example, for a SQL EAI step, select **EAISQL** - **SQL EAI step plug-in**.)

   You must select an entry when you first create the step; it cannot be changed later. The list box displays EAI step types that are available as client EAI plug-ins. This name is used as the link between the EAI step and the run-time plug-in registered on the iProcess Engine(s).

> **Note:** If the appropriate EAI step does not appear in the drop-down list, you must ensure the plug-in is installed using the **Options > Install iProcess plug-in** menu option. (For installation instructions, see the appropriate EAI plug-in installation documentation.)

5. (Optional) On the **Options** tab, select the **Don't delete work items on withdraw** option. If this option is selected, and the deadline on an outstanding step expires or it is withdrawn as an action (release or deadline expire) of another step:
   - the deadline actions are processed.
   - the step remains outstanding (the step remains in the workqueue or the sub-procedure case is not purged).
   - when the step is released (or the sub-procedure case completes) the normal release actions are not processed but the case field data associated with the release step (e.g. the field values set in a normal step whilst in a work queue or the output parameters of a sub-case) is applied to the main case data.
6. (Optional) Click the **Ignore Case Suspend** check box if you want the EAI step to still be processed as normal while a case is suspended by a TIBCO iProcess Objects or SAL application.

   If **Ignore Case Suspend** is not checked (the default option), the EAI step is not processed while the case is suspended. This means that:
   - work items generated by the EAI step are marked as unavailable and cannot be opened (until the case is re-activated.)
   - deadlines on work items generated by the EAI step are not processed. The date and time at which deadlines are due are not affected, and deadlines continue to

expire. However, no actions are processed when a deadline expires. When the case is re-activated, any expired deadlines are immediately processed.

Note: Cases can only be suspended and re-activated from a TIBCO iProcess Objects or SAL application. Audit trail messages indicate whether a case is active or suspended. Refer to the TIBCO iProcess Objects documentation for more information about suspending cases.

7. Click:

   the **Deadlines** tab if you want to enter deadline information for this step. (For an explanation of defining deadlines, see “Defining Deadlines” in the *TIBCO iProcess Modeler - Basic Design* guide.)

   the **Audit Trail** tab to define custom audit trail entry expressions.

   the **Delayed Release** tab to define delayed release settings.

8. On the **Definition** tab, click **EAI Call-Out Definition**.

Note: The name of this button may be different depending on the **EAI Type** you have selected.

9. A dialog specific to your chosen EAI step type is displayed. This allows you to enter the specific information required by your chosen EAI step type.

   The layout of this dialog box and the information you need to supply differ according to the type of EAI step. For more information, either:

   — click the **Help** button in the step type dialog box.

   — choose the appropriate entry for the step type from the **Help > Plug-ins** menu.

   For example, if you want to define a SQL EAI step:

   Select **EAISQL** - **SQL EAI step plug-in** from the **EAI Step Type** drop-down list. The **EAI Call-Out Definition** button changes to **Edit Stored Procedure Call-Out**, and is enabled.

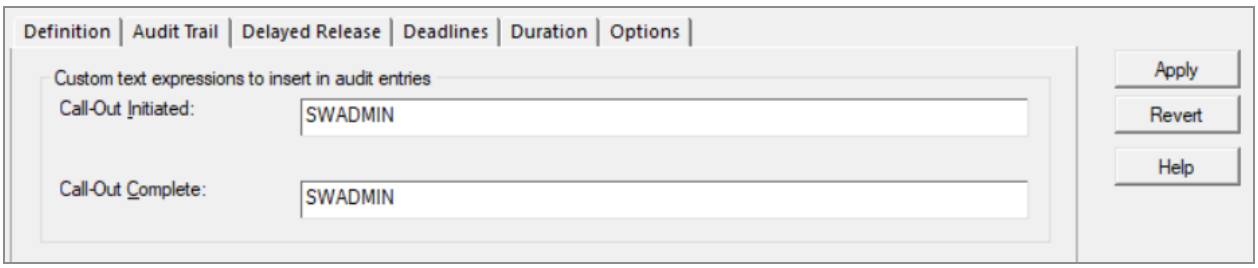
   Click **Edit Stored Procedure Call-Out**. The **SQL EAI Step** dialog is displayed.

Custom Audit Trail Entry Expressions

Click the **Audit Trail** tab to define custom audit trail entry expressions. You can define text expressions that are evaluated when the step is processed and inserted as the %USER value in the audit trail entries.

You must either enter a value in both of the following fields, or leave them both empty:

- In the **Call-Out Initiated** field, enter a valid text expression that replaces the %USER value in the audit trail when the call out is initiated.
- In the **Call-Out Complete** field, enter a valid text expression that replaces the %USER value in the audit trail when the call out is complete.

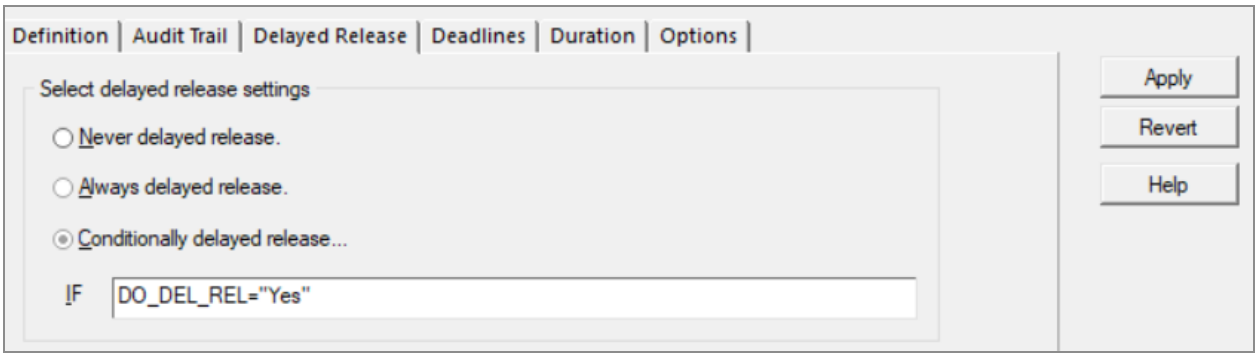


Delayed Release Settings

Click the **Delayed Release** tab to define delayed release settings.

Select one of the following options:

- **Never delayed release** - The step is never set to delayed release so it is released immediately.
- **Always delayed release** - The step is released by the external application.
- **Conditionally delayed release** - The step is delayed release if a specific condition expression evaluates to True. If you select this option, you need to enter a valid condition expression following the IF statement. For example, IF DO_DEL_REL= “Yes”.



Note: The run-time plug-in on the server can override these settings if it returns a status of Delayed Release when the step is processed. For more information about how the plug-in interfaces operate, see the *TIBCO iProcess Plug-in SDK User's Guide*.

Using Transaction Control Steps

Transaction Control (TC) steps can be used to control how transactions are processed for multiple, sequential EAI steps in your procedure flow. By default, the iProcess Background process groups a series of connecting EAI steps and processes into one transaction. TC steps can be inserted in the procedure to break the transaction into multiple transactions.

Overview of Transactions and Transaction Control Steps

A transaction is one unit of work. The iProcess Background process carries out this unit of work. It uses messages that are stored in repositories called Mboxes. Mboxes provide the communication link between iProcess Workspace (Windows) and the iProcess Engine.

A transaction is when the iProcess Background process:

- reads the message from an Mbox
- processes a number of steps
- commits the resulting data back to the database.

This represents one transaction. For more information, see "Mbox Sets and Message Queues" in *TIBCO iProcess Engine Architecture Guide*.

The iProcess Background process bundles all consecutive or concurrent EAI steps into the same transaction. If there is a failure in one step then all steps are rolled back.

TC steps can be used to break up sequences of EAI steps into multiple transactions.

Why Use TC Steps?

There are two reasons for using TC steps in your procedure:

- To have more control over business processes.

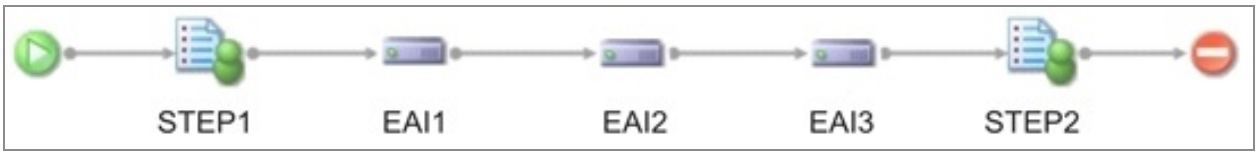
When using a series of EAI steps in your procedure flow, the entire processing of those EAI steps is performed as one transaction. In some situations, you may want more granularity in how the transaction is processed. By breaking up the transactions into multiple transactions, a failure on one EAI step does not mean rolling back a long sequence of EAI steps. Instead, you can break up the sequence of EAI steps so that only a few are rolled back.

- To abort a transaction under control of a procedure.

  Abort TC steps can be used in procedures where there is a mix of transactional and non-transaction EAI steps. This is useful because when non-transactional steps are rolled back, they are just re-tried. By having the procedure check for an error, it can execute a correction activity for the non-transactional step and then use an aborting TC step to fail the whole transaction and rollback both the transactional and non-transactional EAI steps. When the whole transaction is tried again, the problem with the non-transactional step is now corrected, so the procedure progresses successfully.

Examples of TC Step Usage

In the following example, the release of STEP1, the execution and release of all the EAI steps and the sending out of STEP2 is included in a single transaction. A failure of any one of the EAI steps would result in the whole transaction being rolled back to STEP1:



TC steps separate transactions in your business process. There are four situations where TC steps can be used:

- To break up a long sequence of EAI steps
- To separate branches
- To separate parallel branches
- To abort a transaction under control of a procedure

Breaking up a Long Sequence of EAI Steps

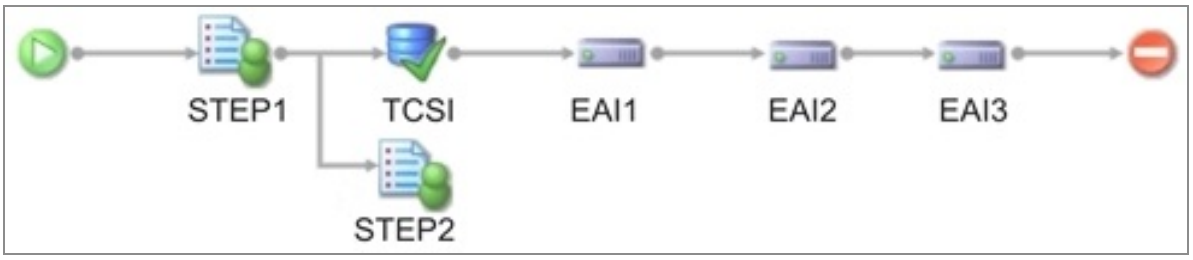
The following example illustrates how a TC step can be used to break up a long sequence of EAI steps.

By adding a TC step before EAI2, the first transaction consists of the release of STEP1 and the execution and release of EAI1. The second transaction is the execution and release of EAI2 and EAI3 and the sending out of STEP2. This means that if a failure occurred in the remaining EAI steps (EAI2 and EAI3), the business process is only rolled back to EAI1.

You can use a **Commit and Continue** or a **Commit and Concede** TC step in this situation. See Types of TC Steps.

Separating Branches

The following example illustrates how to use a TC step to separate branches of EAI steps.

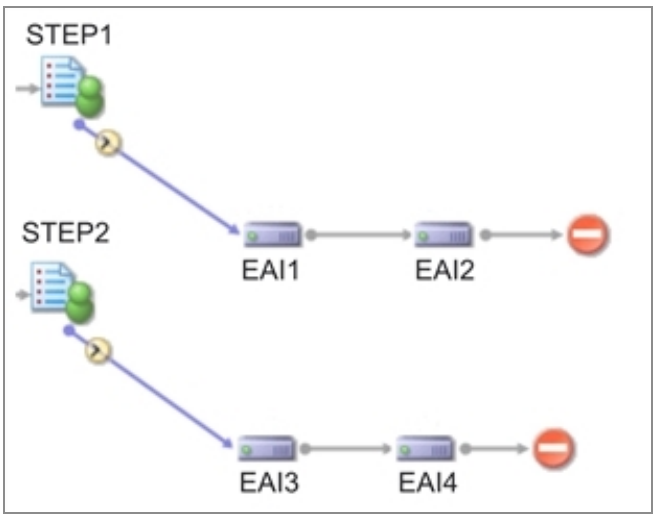


By adding a TC step after STEP1, the first transaction is the release of STEP1 and the sending out of STEP2. The second transaction starts with the execution of EAI1 and concludes with the release of EAI3. This means that if a failure occurs within the EAI steps in the first branch, only the EAI steps are rolled back. STEP2 is not rolled back.

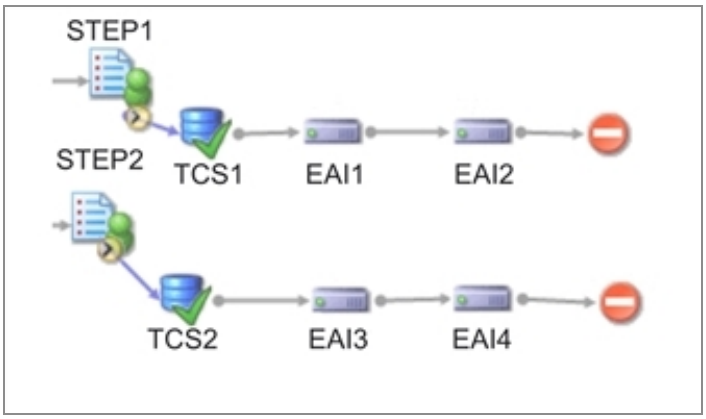
You can use a **Commit and Continue** or a **Commit and Concede** TC step in this situation. See Types of TC Steps.

Separating Parallel Branches

In the procedure shown below, if the deadlines on STEP1 and STEP2 expire at the same time, all of the EAI steps are processed in the same transaction. This means that if one of the EAI steps fail, every EAI step in the parallel branches are rolled back.



The procedure below illustrates how adding a TC step after STEP1 and STEP2 can separate the parallel branches so that each branch is processed as a separate transaction.



You can use a **Commit and Continue** or a **Commit and Concede** TC step in this situation. See Types of TC Steps.

Note:

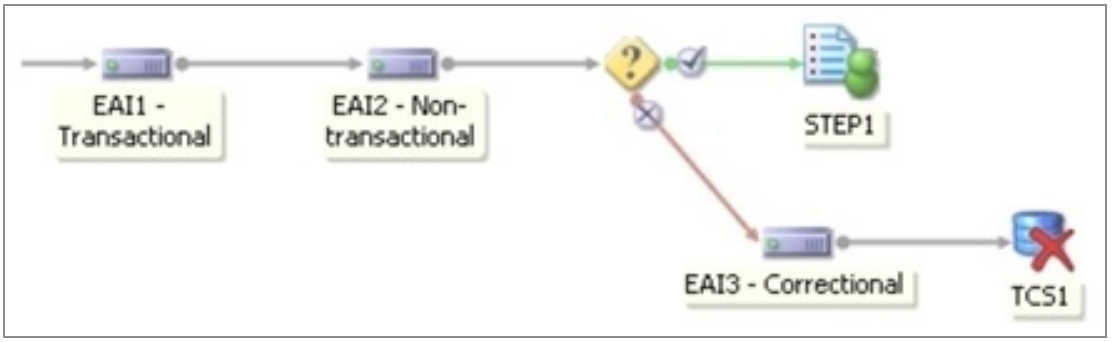
- Within pairs of transactions, the transaction that runs last views data from the first. However, you cannot predict the order in which the parallel paths are processed. This means that you cannot determine which transaction sees the data first. Therefore, if you are using parallel paths, you must decide whether you want these steps to be processed as separate transactions or within one transaction.
- When joining paths into a TC step, you can should use a Wait step.

Defining a Procedural Abort of a Transaction

Using TC steps you can now define an abort TC step in a procedure. The procedure below illustrates how to use a TC step to abort a mixture of transactional and non-transactional EAI steps. This is useful because when non-transactional steps are rolled back, they are just re-tried. By using an abort TC step you can build into the procedure a method for handling errors so that a transaction can be re-tried without having to re-process the case.

In the procedure below, if the condition is not met EAI3 carries out a correction activity on non-transactional EAI2. This happens outside of the transaction. Then TCS1 aborts the whole transaction and rolls the procedure back to EAI1. The next time the transaction is tried it should progress successfully because the problem with non-transactional EAI2 has been resolved.

Warning: An Abort TC Step must always be placed immediately after a step whose type is EAI.



You must use an **Abort Transaction** TC step in this situation. See Types of TC Steps.

Types of TC Steps

When you define a TC step, you can choose how it behaves. A TC step can behave in one of the following ways:

- **Commit and Continue** - choose this option if you want to commit the current data at the current point in the business process and start a new transaction for subsequent steps using the same iProcess Background process. The benefit of choosing this option is that it is fast, as the same Background process starts the new transaction.
- **Commit and Concede** - choose this option if you want to commit the current data at the current point in the business process and start a new transaction for subsequent steps, but using a different iProcess Background process. The iProcess Background process completes the first transaction and updates the database. It then sends a message back to the Mbox where the messages are stored. Processing continues when the iProcess Background process (either the same one that processed the first transaction or another one) reads the message from the Mbox and processes it. The benefit of choosing this option is that it enables load balancing because a different iProcess Background process can process the second transaction. The **Commit and Concede** method is also required if you are using a Tuxedo environment.
- **Abort Transaction** - choose this option to abort an EAI step in an iProcess procedure. This option is always used with an iProcess Condition. A TC step that is configured with the Abort method must always follow an EAI step. It cannot follow any other type of step. For an example of a procedure that uses this method, see Defining a Procedural Abort of a Transaction.

Defining a Transaction Control Step

To define a transaction control EAI step in your procedure:

1. In the **TIBCO iProcess Modeler**, click the Transaction Control EAI Step tool  then place it on your procedure definition.
2. The Properties pane is displayed.
3. Enter the **Name** and **Description** for the step.
4. You now need to decide what type of TC step you want to define:

— Commit and continue - Defining a Commit and Continue TC Step

— Commit and concede - Defining a Commit and Concede TC Step

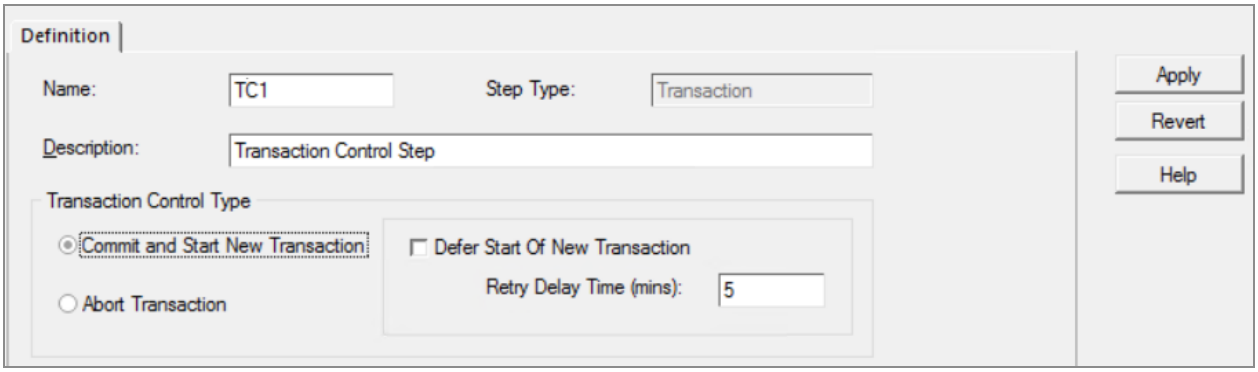
— Abort Transaction - Defining an Abort TC Step.

For more information about the differences between the TC types, see Types of TC Steps.

Defining a Commit and Continue TC Step

After completing the general TC step definition, complete the following steps to create a **Commit and Continue TC** step:

1. In the **Transaction Control Type** area of the pane, choose **Commit and Start New Transaction**.



2. You have the option of entering a **Retry Delay Time** value (in minutes). The default value is set to 5 minutes but values between 1 and 10080 are allowed - (1 minute to 1 week). This means if the continuing transaction fails, it will be retried after the time you specified in the **Retry Delay Time** box has elapsed. The benefit of this is that it enables you to handle the behavior of the EAI steps following the transaction control step. For example, the following EAI sequence takes a specific period of time to complete. This means you do not want it to retry too quickly because the EAI sequence may be still going or you may not have had a chance to correct the fault that caused it to fail. Setting the **Retry Delay Time** value enables you to specify an appropriate period for the retry of the failed EAI step.

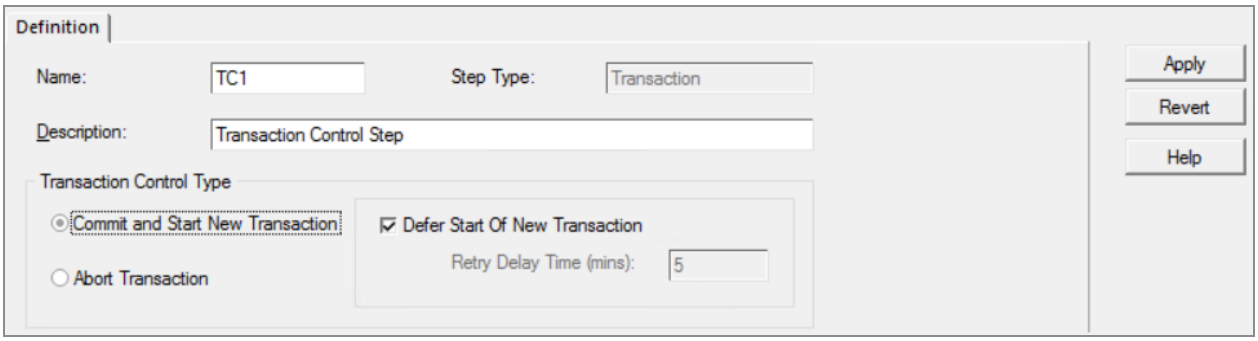
Note: This is the initial retry value only. If the EAI step is re-tried again then the iProcess Background (BG) process uses the `IQL_RETRY_DELAY` process attribute to generate the instructions to retry the EAI step, whose default is 5 minutes. For more information about the `IQL_RETRY_DELAY` process attribute, see *TIBCO iProcess Engine Administrator's Guide*.

3. Click **Apply**.

Defining a Commit and Concede TC Step

After completing the general TC step definition, complete the following steps to create a **Commit and Concede TC** step:

1. In the **Transaction Control Type** area of the pane, choose **Commit and Start New Transaction**.

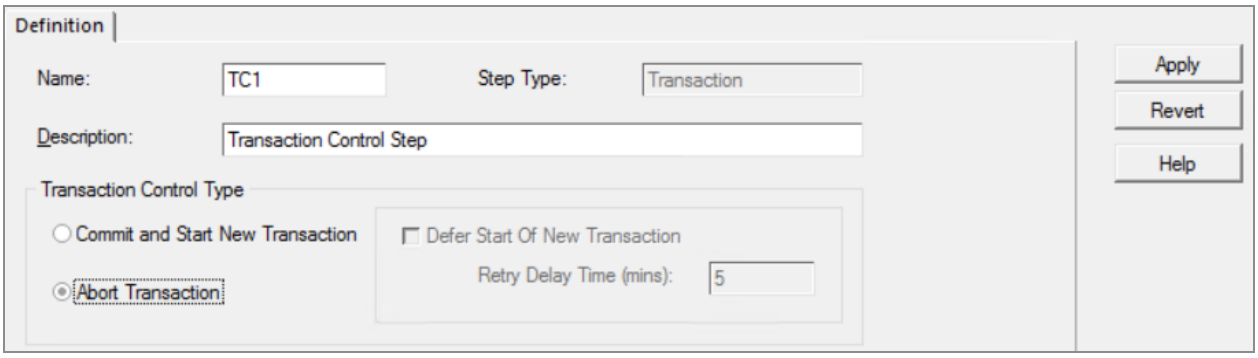


2. Click the **Defer Start Of New Transaction** check box.
3. Click **Apply**.

Defining an Abort TC Step

After completing the general TC step definition, complete the following steps to create an **Abort TC** step:

1. In the **Transaction Control Type** area of the pane, select **Abort Transaction**.



2. Click **Apply**.

Using Graft Steps

This section discusses the use of graft steps in your procedure. Graft steps are used to attach iProcess sub-processes and/or external processes to your main procedure, which have been started by an external application.

Note:

- Graft steps can only be triggered by an external application using TIBCO iProcess Objects calls. For details of how to do this, see the *TIBCO iProcess Objects Client documentation*.
- It is not possible to use graft steps in conjunction with the Jump To feature. If a graft step is named in the send or withdraw lists for the Jump To command, the current transaction gets aborted.

Before using graft steps, you should read the information about array fields and sub-procedure parameter templates because you need to know this before you setup and use graft steps. For more information, see "Using Arrays" and "Defining Sub-procedure Parameter Templates" in *TIBCO iProcess Modeler Advanced Design*.

What is a Graft Step?

A graft step enables an external application to graft (attach) one or more sub-procedures to a particular point in your procedure at run-time. Therefore, when a case of the main procedure is started, the external application can start a number of sub-processes which are attached to the main procedure via the graft step. When all of the required sub-processes have completed, the graft step is released and the case continues.

For example, a financial application determines that a credit check and a transfer of funds are required as part of the main procedure. When another case is started, it determines that only a transfer of funds is required. This means that the procedure is dynamic and cannot be decided at procedure definition time. One of the processes is an iProcess sub-procedure and the other is a process run by the financials system.

How Does a Graft Step Work?

Graft steps are similar to dynamic calls to multiple sub-procedures, but the difference is that the sub-processes are started from an external application and then attached to a specific point in the procedure using the graft step.

When defining a graft step, you have to use a sub-procedure parameter template so that the output mappings are defined for any sub-processes that are started by the application. There are no input parameter definitions required because the external application starts the sub-processes with the case data. This is provided as part of a TIBCO iProcess Objects call sending the sub-procedure fields as name/value pairs.

You place a graft step on your main procedure at the point where you require any external processes or iProcess sub-procedures to be attached. When all the sub-processes have completed, the case data is transferred to the main procedure according to the output parameter mappings. Even if a non-iProcess process has been started by the application, the graft step remains incomplete until that process has completed.

iProcess uses a task count as the protocol between the external application and the TIBCO iProcess Engine to determine how many processes are associated with a graft step. When the task count has been defined by the external application, iProcess knows how many processes need to be completed before the graft step can be released. For more information about the task count, see Graft Step Task Count.

Graft Step Task Count

For a graft step to complete and its release actions to be processed, iProcess needs to know that the processes that are attached to the graft step are complete. To do this, the external application sets the number of tasks that it attaches to the graft step and communicates this to iProcess using the **SetGraftTaskCnt** TIBCO iProcess Objects method and **TaskCnt** property. This pre-defines how many tasks have to be completed before the graft step can be released by iProcess. A task can be made up of cases of a sub-procedure or external processes. For each task, the external application can specify that:

- 0 or more sub-cases can be started and grafted to the graft step or,
- 0 or more non-iProcess external processes can be grafted to the graft step for each task or,
- a mixture of both can be started and grafted to the graft step.

For example, if a funds procedure sends a request to the financial system to carry out an audit funds process, the task count is set to 1. If the request also requires a process to track the funds, the task count would be set to 2.

However, the financial system may have three processes to run as part of task 1 such as debit, credit and audit. These are grouped under the task count of 1. When the task is complete i.e. all the processes have completed, the external application decrements the task count automatically by 1 via SPO. Each time the **StartGraftTask** or **DeleteGraftTask** SPO function is called, the task count (**TaskCnt**) is decremented by 1. If the second task, resulted in no process needing to be run, a delete task operation using the **DeleteGraftTask** function decrements the task count by 1.

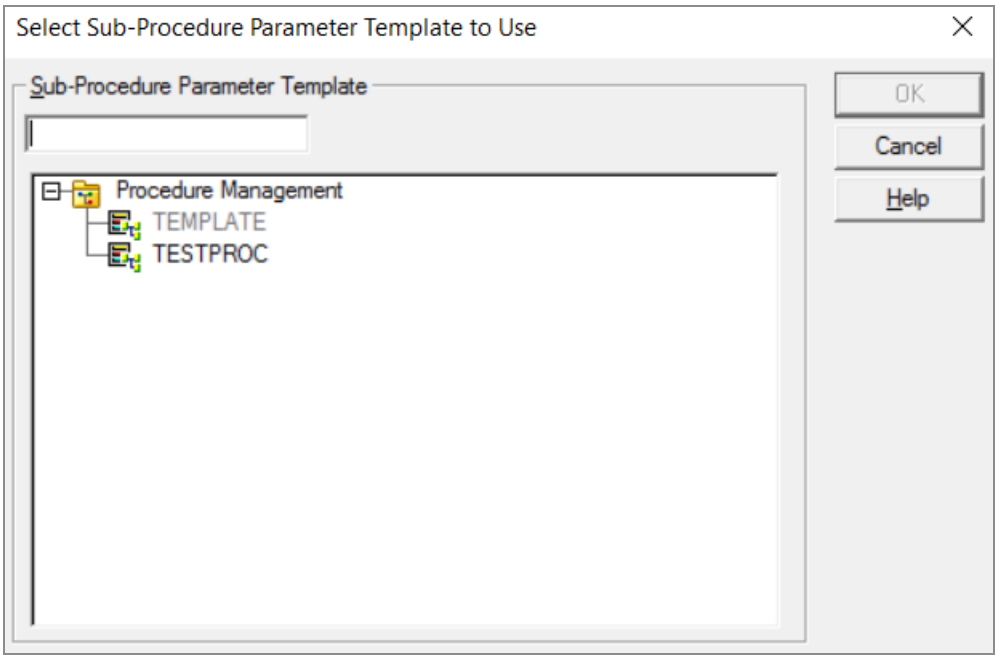
The graft step is complete when the task count reaches 0.

Defining a Graft Step

To define a graft step in your procedure:

1. Click the graft step object and place it on your procedure chart.

   The **Select Sub-Procedure Parameter Template to Use** dialog box is displayed.

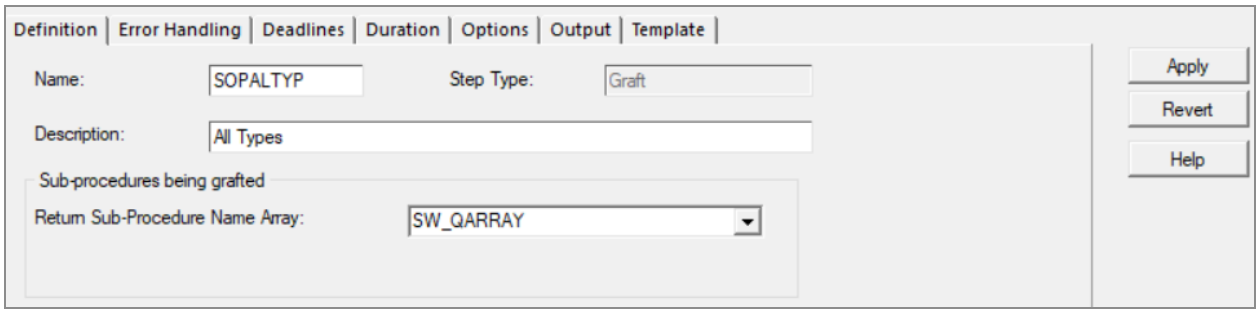


2. Select the template you want to use for the sub-procedure parameters. This defines which sub-procedures you can call from this graft step. For more information about

templates, see "Defining Sub-procedure Parameter Templates" in *TIBCO iProcess Modeler Advanced Design*.

Note: All sub-procedures that are grafted to this graft step must use the same template.

3. The Properties pane is displayed showing the **Definition** tab.



4. Enter a name (up to 8 characters) for the graft step in the **Name** field.
5. (Optional) Enter a description (up to 24 characters) for the call in the **Description** field.

Note: These names do not have to match the sub-procedure's name and description as they are only for use in this procedure.

6. In the **Return Sub-Procedure Name Array** field, select an array field that is used to list both the sub-procedures and external processes that have been grafted. For more information about arrays, see "Using Array Fields" in *TIBCO iProcess Modeler Advanced Design*.

Note: This array field must have been previously defined in your procedure as a text field. The procedure has to be set up so that the array field can be populated with the names of the sub-procedures that need to be started before this graft step call is made.

7. On the **Options** tab, click the **Ignore Case Suspend** check box if you want the step to be processed as normal while a case is suspended by a TIBCO iProcess Objects or SAL application.

   If **Ignore Case Suspend** is not checked (the default option), the step is not processed while the case is suspended.

Note: Cases can only be suspended and re-activated from a TIBCO iProcess Objects or SAL application. Audit trail messages indicate whether a case is active or suspended. For more information about suspending cases, see *TIBCO iProcess Objects Programmer's Guide*.

8. (Optional) On the **Options** tab, select the **Don't delete work items on withdraw** option. If this option is selected, and the deadline on an outstanding step expires or it is withdrawn as an action (release or deadline expire) of another step:
   - the deadline actions are processed.
   - the step remains outstanding (the step remains in the workqueue or the sub-procedure case is not purged).
   - when the step is released (or the sub-procedure case completes) the normal release actions are not processed but the case field data associated with the release step (e.g. the field values set in a normal step whilst in a work queue or the output parameters of a sub-case) is applied to the main case data.
9. (Optional) Click the **Error Handling** tab to define the error handling conditions that apply to this graft step. For more information, see Troubleshooting Graft Steps.
10. (Optional) Click the **Deadlines** tab if you need to define a deadline for this graft step. For more information, see "Defining Deadlines" in *TIBCO iProcess Modeler Basic Design*. If you are using deadline withdrawals on graft steps, see Using Deadline Withdrawals on Graft Steps.
11. Click the **Output** tab to define the output parameter mappings. See Defining Output Parameter Mappings.

Note: Only output parameter mappings need to be defined because the input parameters are provided by the external application calling the graft step (via SPO calls).

12. (Optional) Click the **Template** tab if you want to check which template is currently associated with this graft step call. You can also select a different template to associate to this call.

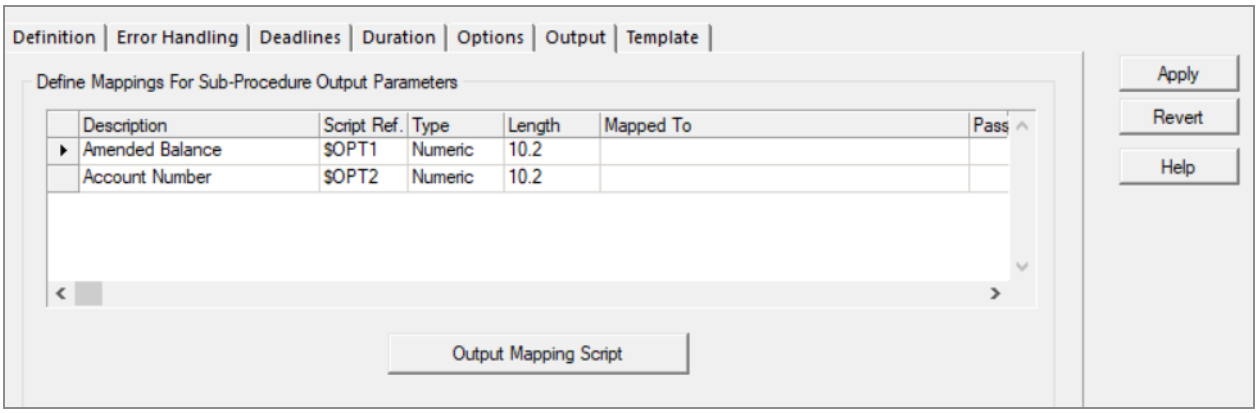
Note: If you do change templates, you will need to remap all of your parameter mappings for dynamic sub-procedure calls and graft steps that use this template.

Defining Output Parameter Mappings

When defining the graft step, the Properties pane enables you to select the sub-procedures you want to start and then define the output mappings. To define the output mappings, perform the following steps:

1. On the Properties pane, click the **Output** tab.

   A list of the pre-defined sub-procedure output fields is displayed. You need to map these to fields in the main procedure. You can also create a private script that you can execute after the mappings have been processed to further manipulate the output values.



2. Select a sub-procedure parameter and then select the main procedure field to map it to from the **Mapped To** column. Only fields of the same type as the one selected are displayed e.g. numeric or text.
3. Click **Apply** to save your settings.

Running an Output Mapping Script

To provide extra manipulation of output values, you have the option to execute a private script on completion of the output mappings. The script can refer to the sub-procedure output values using keywords with the format $OP*n* or $OPT*n* where:

- *n* is a positive integer that is automatically assigned by iProcess to each output parameter.
- T denotes that the parameter has been inherited from a template.

The script can also refer to, and assign to, the main procedure fields. For more information about creating private scripts, see "Using Scripts" in *TIBCO iProcess Modeler Advanced Design*.

Withdrawing a Graft Step

If you have steps that become redundant during the running of a case, you can define the procedure so that those steps are withdrawn from the work queues. You do this by defining a withdraw action on the step. However, if you want to withdraw a graft step in this way, you must ensure that **all** of the following pre-requisites are met before doing so:

- An external application must inform the graft step how many tasks it needs to complete (using the **SetGraftTaskCnt** SPO method and **TaskCnt** property).
- The graft step's task count (**TaskCnt**) must reach zero (the task count is decremented when a task has completed or if you delete a task).
- The previous step has been released, causing the graft step to be processed.

Note:
Withdrawing a graft step has the following effects:

- All outstanding sub-procedures are immediately closed. This is recorded in the audit trail.
- iProcess stops processing along that part of the procedure definition branch.

This means that even if a graft step receives the withdraw notification, it is not withdrawn until such time that the task count is received and enough tasks are grafted (or decrement task counts called) to end up with a task count of 0.

Using Deadline Withdrawals on Graft Steps

Because a graft step is closely linked with an external application, please note the following points when using a deadline with a withdraw on your graft step:

- The withdraw can only happen when all required graft step elements have been received i.e. the previous step has been released, the graft count is set and the correct number of tasks have been grafted. A withdraw audit trail entry is not displayed until the above elements have been completed.
- This means that even if the graft step has started and the sub-procedures have started, you do not get outstanding sub-procedures when the graft step withdraws because the entire graft step is still outstanding.

- Once the graft step has been withdrawn, iProcess stops processing along that part of the procedure definition branch.

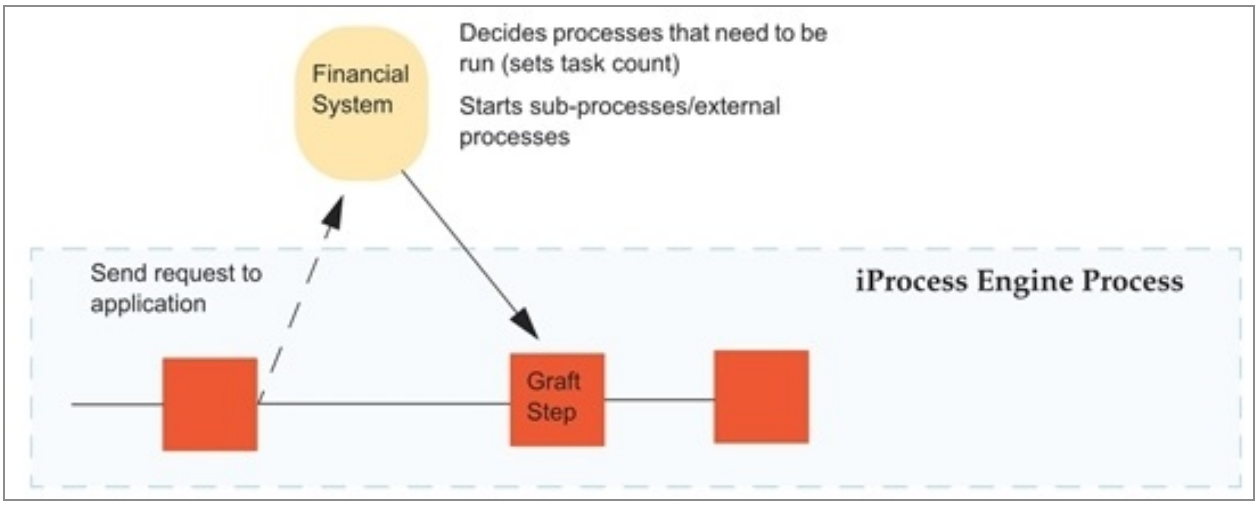
Example of Using a Graft Step

The following example demonstrates the use of a graft step to attach several processes to your iProcess procedure when a case is run. This example is based upon using a financial application as the external application and this triggers the graft step using a TIBCO iProcess Objects call.

Brief

Create a procedure for an online banking, account management procedure. iProcess is used to control the flow of the process but the financial application determines what account processes are run e.g. transfer funds, account balance, pay bill, etc. The application triggers the required processes based upon the details captured by case data in iProcess. Some of the processes are external to iProcess but some are defined as iProcess sub-procedures.

Example Procedure Design



Running a Case of the Procedure

1. A case of the procedure is started by a customer. As the case progresses and details of the account are entered, iProcess sends a request to the financial application so

that it can determine what processes need to be started.

The request consists of a call to the financial system to “manage the account”. The financial system decides what processes need to be run as part of this request and sets the task count based upon the number of processes it needs to run - for example, a get account balance process and an issue new card process is required. The task count is communicated to iProcess via the **SetGraftTaskCnt** method.

2. In this example, a new debit card and an account balance is required by the customer. These two processes are started by the application and grafted to the graft step in the main procedure.
3. The process to order a new debit card is an iProcess sub-procedure while the get account balance process is an external process that is performed by the financial system. The TIBCO iProcess Objects **StartGraftTask** method is used to start the processes and to attach them to the graft step.
4. After the new debit card sub-procedure completes and the external application has marked the get balance process as complete, the return values are passed back to the application using the graft step output mappings. When the task count reaches 0, the graft step is complete and is released. The financial system has to inform iProcess that the external process has been completed before the step can be released.

Troubleshooting Graft Steps

There are a number of ways you can troubleshoot errors with graft steps:

- Set up and inspect the graft step return status codes - see Return Status Codes.
- Define how the process stops if the graft step fails - Stopping the Process if an Error Occurs.

Return Status Codes

You can use the following return status code array field values to help troubleshoot problems with graft steps. If you set up a return status, numeric array field, you can capture a status code to help troubleshoot problems. The return codes you can get are:

Return Status	Description
SW_NA	The starting of the sub-procedure case has not been attempted.
1	The sub-case has been grafted and started successfully.
2	The sub-case has completed successfully.
-1	There was an error starting the sub-case: Invalid sub-procedure name.
-2	There was an error starting the sub-case: call step and sub-procedure use different parameter templates.
-3	N/A (only applies to dynamic sub-procedure call steps.)

Stopping the Process if an Error Occurs

The graft step call definition provides parameters that you can use to stop a case of your process (at the graft step) if a specific error occurs. If these are not selected, the sub-cases and process continue but you may have errors in the case data. On the **Graft Step** pane, click the **Error Handling** tab and choose one or more of the following options:

- **Sub-procedure names that are invalid**

  Select this option to stop the process if iProcess cannot find one of the sub-procedures it needs to call.

- **Sub-procedures that do not use the same sub-procedure parameter template**

  Select this option to stop the process if iProcess finds parameters that are not in the sub-procedure parameter template being used by the graft step.

- **Sub-procedures that use different versions of the same sub-procedure parameter template**

  Select this option to stop the process if iProcess finds some parameters that are not valid for the version of the template being used for this call.

  For more information about versions of procedures and templates, see “Using Version Control” in the *TIBCO iProcess Modeler - Procedure Management* guide.

Using Public Steps & Events

This is a feature that enables iProcess to publish information about case start and event trigger steps to an external process (or application) via SAL or TIBCO iProcess Objects interfaces.

To do this, you specify steps or events in a procedure which you want to be public (via the TIBCO iProcess Modeler). The result is a subset of events or steps that are now available to external processes or applications.

For each public step or event, you can specify a list of expected input fields and define the URL of the document describing the purpose of the public step.

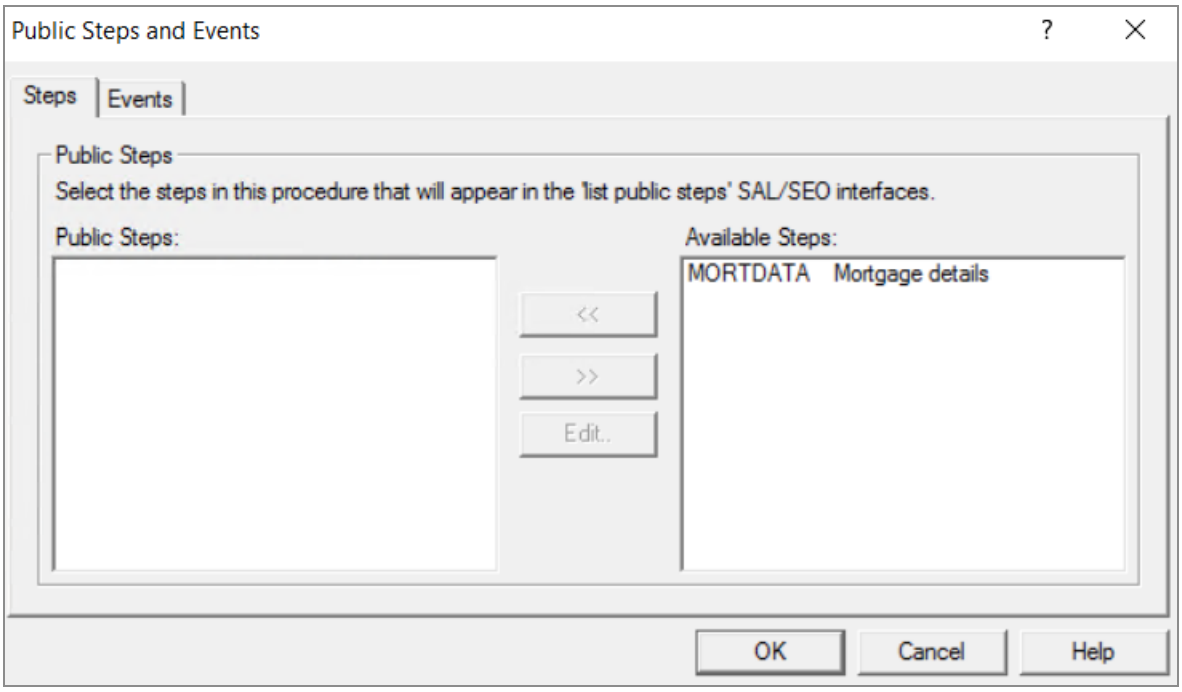
Note: This feature is provided so that an external process (or application) can limit itself to perform case start at step or trigger events to only those defined as Public Steps or Events. The public steps or events are not enforced by iProcess or TIBCO iProcess Objects. (This also affects sub-procedures. The list of steps displayed in the **Select Step to Start Sub-procedure At**: can be restricted to those steps defined as Public steps. For more information about sub-procedures, see *TIBCO iProcess Modeler Advanced Design*).

Defining Public Steps

To create a new public step, perform the following steps:

1. In the TIBCO iProcess Modeler, ensure you have your procedure open and then click **Procedure** > **Public Steps...**.

   The **Public Steps and Events** dialog box is displayed.



2. The **Steps** pane shows the available steps in the procedure (as per the last save of that procedure) on the right. The steps shown are those that match the type of step that can be published as case start or close case public steps. It is sorted by step name.
3. Click the step you want to publish, then click  .

   The **Step/Event Properties** dialog box is displayed.
4. Enter the **Public Description** and **Usage URL** for the public step. Note that:
   - The information the external process (or application) sees is the **Step Name**, the **Public Description**, the **Usage URL**, and the list of available fields.
   - If you are working on a sub-procedure, the list of available fields is grayed out for all step types except Event steps. You should use IO parameters if you need to restrict the list of fields available when mapping sub-procedure fields - for more information about defining IO parameters, see *TIBCO iProcess Modeler Advanced Design*.
   - The **Public Description** defaults to the current Step's Description but you can change this if you wish.
   - You can use the **Usage URL** to point to a web page or a local file. For example, **\\www.abc.com\webpage.htm** or **file:///c:/usage.txt**. Alternatively, you can use this field to enter text to contain more information about the public step. This is useful to provide information about how the step is to be used.

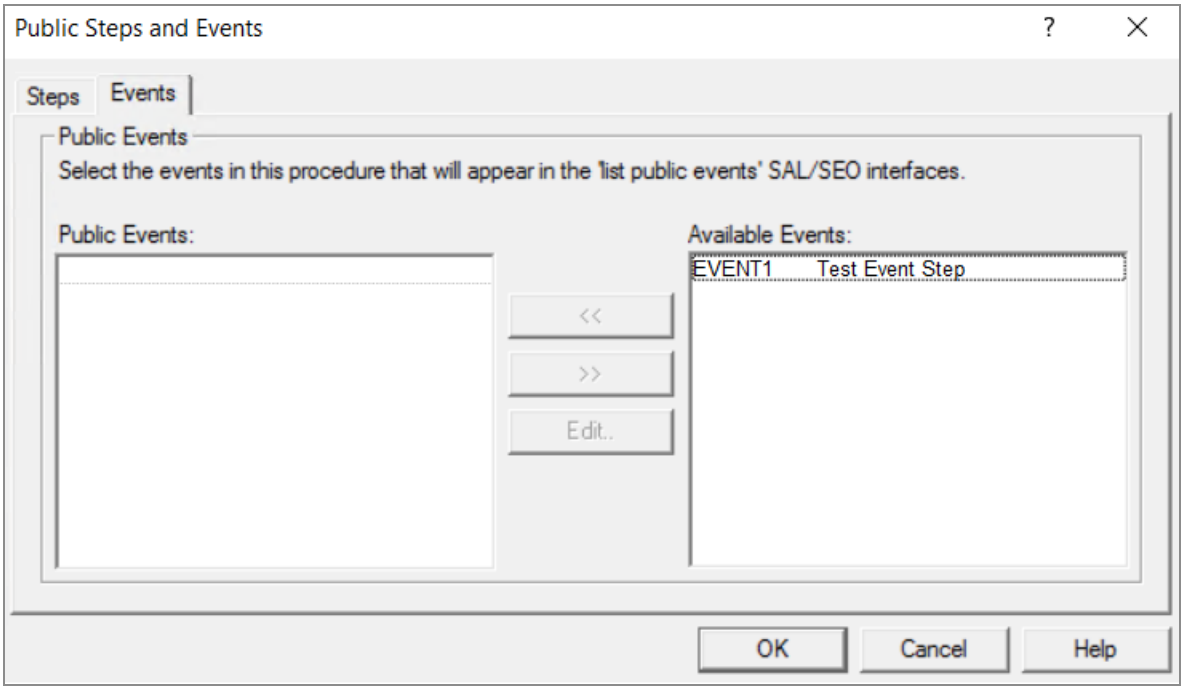
Defining Public Events

To create a new public event, perform the following steps:

1. In the TIBCO iProcess Modeler, ensure you have your procedure open and then click **Procedure** > **Public Steps...** .

   The **Public Steps and Events** dialog box is displayed.

2. Click the **Events** tab. The **Events** pane shows the available events in the procedure (as per the last save of that procedure) on the right. The events shown are those that match the type of event that can be published. It is sorted by event name.



3. Click the event you want published and then click <<.

4. The **Step/Event Properties** dialog box is displayed. Enter the **Public Description** and **Usage URL** for the public event. Note that:

   — The information the external process (or application) will see is the **Event Name**, the **Public Description**, the **Usage URL** and the list of available fields.

   — The **Public Description** defaults to the current Event's Description but you can change this if you wish.

   — You can use the **Usage URL** to point to a web page or a local file. For example, **\\www.abc.com\webpage.htm** or **file:///c:/usage.txt**. Alternatively, you can use

this field to enter text to contain more information about the public step. This is useful to provide information about how the step is to be used.

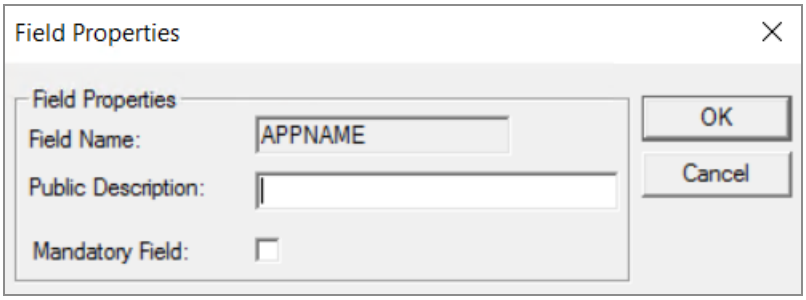
Defining the Field Data

To define the list of fields for the currently selected public step or event:

1. The **Field Data** pane in the **Step/Events Properties** dialog box shows the available fields in the procedure (as per the last save of that procedure) on the right. It is sorted by field name.

> **Note:** If you are defining Public Steps for a sub-procedure, you can only define public fields for Event steps. All other step types have the list of fields grayed out.

2. From the **Available Fields** box, click the field you want to add to the public step/event and then click <<.
3. The **Field Properties** dialog box is displayed.
4. Enter the **Public Description** and if you want the field to be mandatory, select the **Mandatory Field** check box. Note that:
   - The **Public Description** is blank by default. You can enter a description or leave it blank.



   - Select the **Mandatory Field** check box if the user must complete this field before the step can be released.

Importing a iProcess 2000 Procedure

Note: Use the **swutil** utility to import procedures. For more information about using the **swutil** utility to import procedures, see *TIBCO iProcess swutil and swbatch Reference Guide*.

There are two elements to a sub-procedure:

- a sub-procedure call
- a sub-procedure definition

When you import a iProcess 2000 procedure with a sub-procedure into iProcess Version 10.6, it will remain in the iProcess 2000 format. Therefore, to take advantage of the public steps/events feature you have to change the iProcess 2000 sub-procedure into iProcess Version 10.6 format. To do this:

1. Use the iProcess Modeler to open the sub-procedure in iProcess Version 10.6 and save it. The iProcess 2000 sub-procedure is saved in the new format.
2. Open the parent procedure in iProcess Version 10.6, delete the existing sub-procedure call and define a new one. For more information about defining procedures, see *TIBCO iProcess Modeler Advanced Design*.

Note: If you make any changes to the iProcess 2000 sub-procedure in iProcess Version 10.6, you have to change the sub-procedure call in the parent procedure. A iProcess 2000 sub procedure call does not work with an iProcess Version 10.6 sub-procedure definition.

Using TIBCO Formflows

This section describes how to use TIBCO formflows with the iProcess Modeler.

Overview

This feature enables you to develop formflows in TIBCO FormBuilder and then call them from your procedure. This means that when a case of a procedure is run, a formflow can be displayed instead of a work item and vice versa. For more information about developing formflows, see *TIBCO BusinessWorks FormBuilder User's Guide*.

There are two possible scenarios when using formflows:

- a work item can open a formflow. You can configure a step in your procedure to call a formflow. The formflows are displayed in the iProcess Workspace (Browser) at run-time. This means that when a case of that procedure is run, the iProcess Engine passes the formflow definition to the iProcess Workspace (Browser). The iProcess Workspace (Browser) displays the formflow to the user.
- a formflow can open a work item. At run-time, a formflow can display a work item from a case of a procedure in iProcess Workspace (Browser). For more information, see *TIBCO iProcess Workspace (Browser) User's Guide*.

For each of these scenarios, you need to define a step with a form type of **Formflow Form (FORMFLOW)** in your procedure. For more information, see Defining a Formflow in a Procedure.

Note:
This section does not describe:

- the run-time operation of TIBCO iProcess Workspace (Browser). For more information, see *TIBCO iProcess Workspace (Browser) User's Guide*.
- integrating work items with a formflow in TIBCO FormsBuilder. For more information, see *TIBCO iProcess Workspace (Browser) User's Guide*.

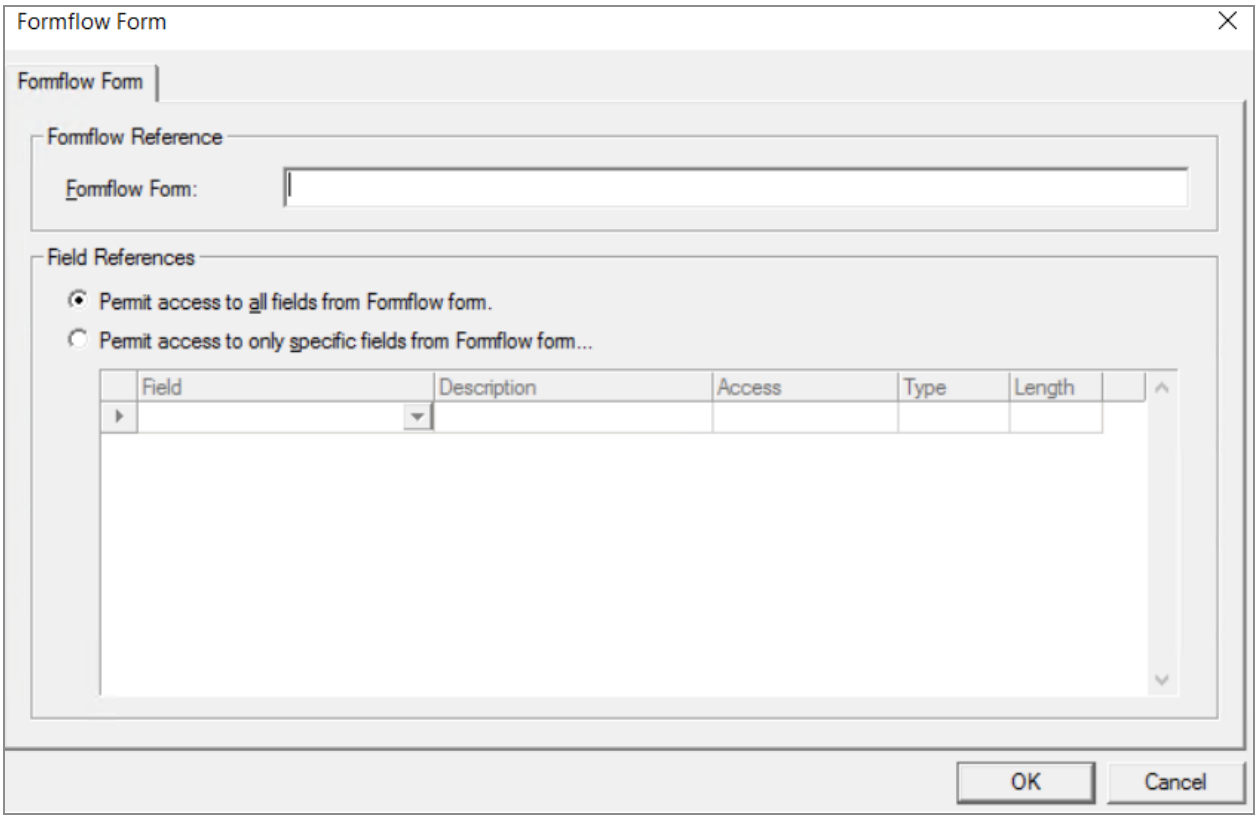
Defining a Formflow in a Procedure

To use formflows with your procedure, you must define a form that has a form type of **Formflow Form (FORMFLOW)** in your procedure. How you define the formflow form depends on the scenario that you want to use:

- if you want a formflow to open a work item. When defining a step that opens a formflow, you must define a step whose form type is **Formflow Form (FORMFLOW)**. This is so that you can specify which procedure fields, if any, can be passed between the formflow and the procedure.
- if you want a step to open a formflow. As well as identifying the form type of the step as **Formflow Form (FORMFLOW)**, you must put in a reference to the specific formflow that the step is to launch. You can then specify which procedure fields, if any, can be passed between the procedure and the formflow.

To define a formflow in your procedure, perform the following steps:

1. Define a step in your procedure. For more information, see "Defining a step" in *TIBCO iProcess Modeler Getting Started*.
2. From the Properties pane, select **Formflow Form (FORMFLOW)** from the **Form Type** drop-down list.
3. Click **Edit**. The **Formflow Form** dialog box is displayed.



4. (Optional) In the **Formflow Form:** box, enter the reference to the formflow that you want to display when a case of this procedure is started. The reference is in the format *formflowrurl\formflow* where:

   — *formflowurl* is the URL of your formflow application.

   — *formflow* is the name of the actual formflow you want to display when a case of this procedure is run.

   For more information, see *TIBCO BusinessWorks FormBuilder User's Guide*.

Note: If the work item is only going to be accessed from a formflow, you do not need to enter a reference. You only need to enter a reference if you want the procedure to launch a formflow.

5. (Optional) When you have selected the reference to the formflow you want to display, you can select a list of the procedure fields that can be accessed from the formflow. You can specify all or some of the fields. From the **Field References** area of the **Formflow Form** dialog box, select one of the following options:

   — **Permit access to all fields from Formflow form**. The formflow can access all the fields in the procedure.

— **Permit access to only specific fields from Formflow form...**. You can specify which procedure fields the formflow should have access to. The following table describes how to do this:

Field	Description
Field	Select the name of the field that you want the formflow to have access to.
Description	Enter a description for the field. The description can be up to 128 characters long.
Access	Select one of the following options: • **Read Only**. This means the formflow can only read from the field, it cannot write any data to it. • **Optional**. This means the formflow can read from or write to the field, depending on how you have defined your procedure. • **Required**. This means the formflow must write some data back to the field.
Type	Automatically retrieved from the field definition. See "Defining Fields" in *TIBCO iProcess Modeler Getting Started*.
Length	Automatically retrieved from the field definition. See "Defining Fields" in *TIBCO iProcess Modeler Getting Started*.

Click **OK**. The step has been defined with a form type of **Formflow Form (FORMFLOW)** and the following icon is displayed:  .

Editing a Formflow Form Type Step

There are two ways you can edit a step whose form type is **Formflow Form (FORMFLOW)**:

- edit the properties of the step.
- change the form type of the step.

Editing the Properties of a Formflow Form Type Step

To edit the properties of a formflow form step:

1. Right-click on the step and click **Properties**. The Properties pane is displayed.
2. Click **Edit** from the Properties pane, the ***stepname* Formflow Form** dialog box is displayed. Depending on your requirements, edit the properties of the step. For more information, see Defining a Formflow in a Procedure.

Using Open Forms

This section describes how to implement Open Forms, an iProcess feature that enables you to use an external forms application in place of iProcess's own Form window for one or more steps of a procedure.

This facility is particularly useful for maintaining a consistent user interface with other applications already used in the enterprise, or where specialized features are required that are not available in standard iProcess forms, such as special controls or embedded graphics.

Open Forms Overview

When a work item is opened that has an Open form defined, the **Form** window is hidden or aborted and control of the work item's form data is passed to the Open Forms application. The application must then:

- get the necessary iProcess field data and display it to the user.
- when the form is closed, pass any changed data back to iProcess, and inform iProcess whether to keep or release the work item.

About the Open Forms SDK

The Open Forms SDK is a toolkit which you can use to develop an Open Forms application. It is supplied with the TIBCO iProcess Modeler and is currently available for C/C++ and Microsoft Visual Basic (VB).

The Open Forms SDK contains the following components:

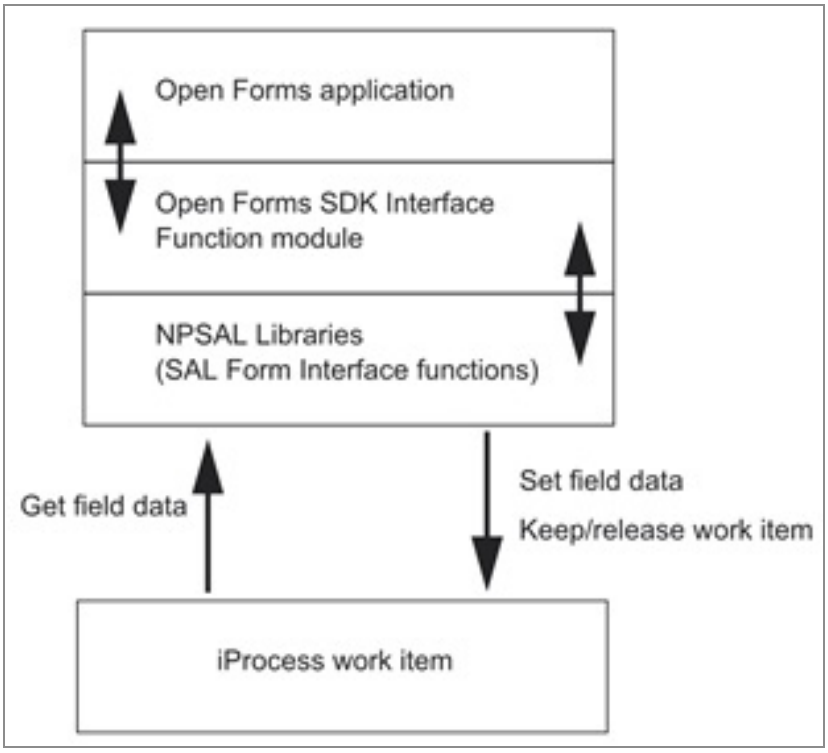
- *Interface Function module*. This is a source code module that you can include in the build of your Open Forms application. It provides interface functions (openform_*xxx*()) which you can call to get and set a work item's field data and keep or release the work item. See Developing the Open Forms Application.

- *Swemail example application*. **Swemail** is a simple application that demonstrates how to implement Open Forms. Full source code, an executable version and an iProcess procedure are all supplied.

  For more information about **Swemail**, see the `Opfrmsdk.pdf` file, which you can find in the `docs` directory on the TIBCO iProcess Modeler distribution CD.

Open Forms and the NPSAL Form Interface

The Open Forms SDK Interface Function module provides simple interfaces which call the iProcess Application Layer (SAL) Form Interface functions in the NPSAL libraries. The SAL interfaces are used to get and set field data and to keep or release a work item.



Using the Open Forms SDK makes it easier to develop an Open Forms application, because you do not need to load the NPSAL libraries and access the SAL Form Interface functions directly.

Note:

- If you want to implement Open Forms using a language for which the Open Forms SDK is not available, you can do so by using the SAL Form Interface directly.
- You can use the Open Form SDK and the SAL Form Interface in the same application if you need to. For example, if you want to access SAL Form interface functions that are not provided by the Open Forms SDK.

For more information about the SAL Form interface, see the `Openuser.wri` file, which you can find in the `docs` directory on the TIBCO iProcess Modeler distribution CD.

How to Implement Open Forms

To implement Open Forms in an iProcess procedure, you must perform the following steps:

1. Develop your Open Forms application, using the Open Forms SDK Interface Function module functions to access work item data.
2. Define an iProcess script to call the Open Forms application. The script must pass the tools window and form session handles to the Open Forms application and hide or abort the iProcess form.
3. Call the script whenever a work item is opened.

The following sections describe these steps in more detail.

Developing the Open Forms Application

Your Open Forms application must perform the following tasks.

Step	Application Task	Relevant Open Form SDK Functions
1.	Load the NPSAL libraries and provide access to the other openform_xxx() functions.	openform_initialise()

Step	Application Task	Relevant Open Form SDK Functions
2.	Provide the user with an appropriate form and populate it with the necessary iProcess data.	Returns
3.	Copy any changed iProcess data back to iProcess when the form terminates.	openform_set_field_value()
4.	Inform iProcess whether it should keep or release the work item.	openform_keep() openform_release()
5.	Unload the NPSAL libraries and terminate access to the other openform_*xxx*() functions.	openform_terminate()

Note: All iProcess field values are passed as **strings**. It is the Open Form application's responsibility to ensure that field values are passed in an appropriate format for the field type.

Creating the Script to Call the Open Forms Application

Note: For more information about creating and editing scripts, see "Creating Scripts" in *TIBCO iProcess Modeler Advanced Design*.

You must create a script, which calls the Open Forms application when the step is opened. The script must do four things:

1. Get the tools window handle, which identifies the iProcess login session that the work item belongs to. You can do this using the GETHANDLE(5) function.
2. Get the SAL form session handle, which identifies the particular work item. You can do this using the GETHANDLE(2) function.
3. Call the Open Form application, passing the tools window handle and SAL form session handles as parameters. You can do this using the WINRUN function.

Note: You could use DDE instead to call the application. See Using DDE to Call the Application.

4. Hide or abort the iProcess **Form** window for this step. You can do this using the FORMCONTROL function:
   - Use FORMCONTROL (4) to hide the window. Any Keep or Release command defined for the step is run when the form is kept or released.
   - Use FORMCONTROL (0) to abort the window. Any Keep or Release command defined for the step does not run when the form is kept or released.

Note: The FORMCONTROL function can be placed anywhere in the step, as it does not take effect until the script has finished.

An example script is shown below. The relevant function calls are pointed out and shown in bold for clarity.

```
; ---------------------------------------------------------------
; Example script to call an Open Form application SWEMAIL.EXE
; ---------------------------------------------------------------
;
; Get the tools window handle.
;

toolshwnd:= gethandle(5)
; Get the SAL form session handle.
;

salformsh := gethandle(2)
; Call SWEMAIL.EXE and pass the tools window handle and SAL form
; session handle.
;

if winrun ("SWEMAIL.EXE" + str(toolshwnd,0) + " " +
str(salformsh,0),1) < 33
     Messagebox ("SWEMAIL","SWEMAIL.EXE failed",4,0)
     formcontrol(2) ; KEEP - put item back in work queue.
     EXIT
endif

; Hide the Form window - any Keep/Release command will be run.
; (formcontrol(0) would abort the Form window.)
;
```

```
formcontrol(4)
EXIT
```

Using DDE to Call the Application

Instead of using WINRUN, you can also call the Open Forms application using DDE. You may want to use DDE if, for example:

- You want to avoid the overhead involved in starting the Open Forms application each time a form is opened.
- You want to run your Open Forms application as a service, continually processing forms as required.
- You want to workflow-enable an existing forms application.

One way of using DDE calls would be:

1. Use DDECONNECT to ‘wake up’ the Open Forms application. (If the call fails because the application is not running you can then start it using WINEXEC.)
2. Use DDEEXECUTE to pass the tools window handle and SAL form session handle to the Open Forms application.
3. Use DDETERMINATE to close the DDE session.

For more information about the available DDE functions, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Calling the Open Forms Application from a Step

To run your Open Form application, call your script as the **Initial** command for the step. (The **Initial** command is run when the step is sent out.)

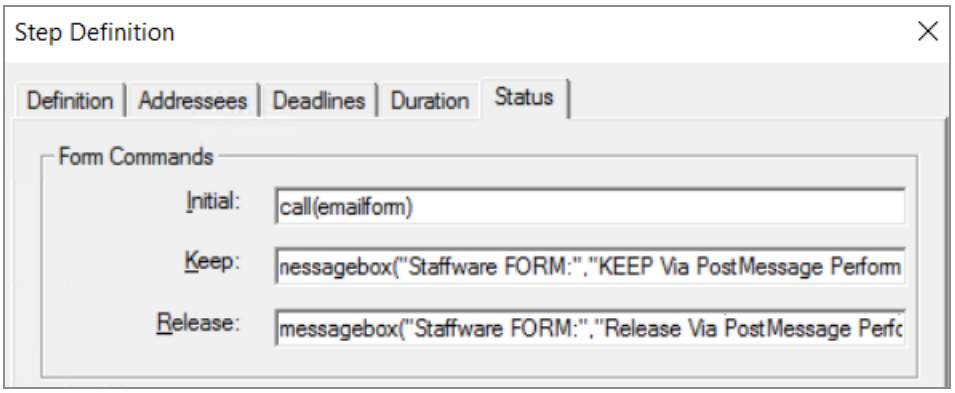
To do this:

1. Click the step that you want to call your Open Forms application from. The Properties pane is displayed.
2. Click in the **Initial** box of the **Form Commands** section.

3. Use the CALL function to call the script. Type:

```
CALL (script)
```

where *script* is the name of your script.



Open Forms Functions

This section describes the functions that are available from the Open Forms SDK Interface Function module. These functions are:

- openform_initialise(). Load the NPSAL support libraries and initialize access to the other functions in the module.
- Returns. Get the value for an iProcess field from a work item's form data.
- openform_keep(). Keep the work item associated with a given form session.
- openform_release(). Release the work item associated with a given form session.
- openform_set_field_value(). Set the value for an iProcess field in a given form session.
- openform_set_recalc(). Enable or disable recalculation of calculated fields and conditional text after a openform_set_field_value() call.
- openform_terminate(). Unload the NPSAL support libraries and terminate access to the other functions in the module.

openform_initialise()

Load the NPSAL support libraries and initialize access to the other functions in this module.

Synopsis

C/C++

```
long int openform_initialise (HWND tools_wnd_hdl, HWND parent_wnd_hdl)
```

VB

```
Function openform_initialise (tools_wnd_hdl As Long, parent_wnd_hdl As Long) As
Long
```

where:

- *tools_wnd_hdl* is the tools window handle that identifies the iProcess login session to which the work item belongs, as passed to your Open Form application by the calling script - see Creating the Script to Call the Open Forms Application.
- *parent_wnd_hdl* is the window handle of your Open Form application window. This will be the parent window of any error message dialogs displayed by the other openform_*xxx* functions.

  If you do not want error messages displayed you should pass *parent_wnd_hdl* as NULL or 0.

Description

You should call this function when your Open Form application first starts up - normally when your main window procedure function receives a WM_CREATE message.

You cannot call any other openform_*xxx* functions until you have called this function.

Returns

One of the following numeric values.

Value	Description
0	(long int) Error.
>0	(long int) *openform_id*. The identifier of the Open Forms SDK Interface Function module, which must be used when calling the other openform_*xxx* functions.

openform_get_field_value()

Get the value for an iProcess field in the given form session.

Synopsis

C/C++

```
long int openform_get_field_value (long int openform_id, long int sal_formsh,
LPSTR lpfield_name, LPSTR lpfield_value)
```

VB

```
Function openform_get_field_value (openform_id As Long, sal_formsh As Long,
lpfield_name As String, lpfield_value As String) As Long
```

where:

- *openform_id* is the identifier of the Open Forms SDK Interface Function module, as returned by openform_initialise().
- *sal_formsh* is the SAL form session handle that identifies the work item, as passed to your Open Form application by the calling script. See Creating the Script to Call the Open Forms Application.
- *lpfield_name* is the name of the iProcess field you want to get the value of.
- *lpfield_value* is a pointer to buffer space where the field value should be returned. This should be large enough to contain the maximum number of characters in the field, plus 1 byte for a NULL terminator.

Description

Use this function to obtain the value for an iProcess field from a work item's form data.

Returns

One of the following numeric values.

Value	Description
1	Success. The field value is returned in *lpfield_value.*
0	Invalid *openform_id.*
-1	Invalid *sal_formsh.*
-2	Invalid *lpfield_name.*
-3	Invalid *lpfield_value.*
-4	np_sal_frm_getdata() interface function not available.

openform_keep()

Keep the work item associated with the given form session.

Synopsis

C/C++

```
long int openform_keep (long int openform_id, long int sal_formsh)
```

VB

```
Function openform_keep (openform_id As Long, sal_formsh As Long) As Long
```

where:

- *openform_id* is the identifier of the Open Forms SDK Interface Function module, as returned by openform_initialise().
- *sal_formsh* is the SAL form session handle that identifies the work item, as passed to your Open Form application by the calling script. See Creating the Script to Call the Open Forms Application.

Description

Use this function when you want to keep a work item (return it to the user's queue).

You must not use the Returns, openform_set_field_value() or openform_release() functions after using this function, because the *sal_formsh* will no longer be valid.

Returns

One of the following numeric values.

Value	Description
1	Success.
0	Invalid *openform_id.*
-1	Invalid *sal_formsh.*
-4	np_sal_frm_getdata() interface function not available.

openform_release()

Release the work item associated with the given form session.

Synopsis

C/C++

```
long int openform_release (long int openform_id, long int sal_formsh)
```

VB

```
Function openform_release (openform_id As Long, sal_formsh As Long) As Long
```

where:

- *openform_id* is the identifier of the Open Forms SDK Interface Function module, as returned by openform_initialise().
- *sal_formsh* is the SAL form session handle that identifies the work item, as passed to your Open Form application by the calling script. See Creating the Script to Call the Open Forms Application.

Description

Use this function when you want to release a work item from a user's work queue and continue the work flow.

You must not use the Returns, openform_set_field_value() or openform_keep() functions after using this function, because the *sal_formsh* will no longer be valid.

Returns

One of the following numeric values.

Value	Description
1	Success.
0	Invalid *openform_id.*

Value	Description
-1	Invalid *sal_formsh.*
-4	np_sal_frm_getdata() interface function not available.

openform_set_field_value()

Set the value for a iProcess field in the given form session.

Synopsis

C/C++

```
long int openform_set_field_value (long int openform_id, long int sal_formsh,
LPSTR lpfield_name, LPSTR lpfield_value)
```

VB

```
Function openform_set_field_value (openform_id As Long, sal_formsh As Long,
lpfield_name As String, lpfield_value As String) As Long
```

where:

- *openform_id* is the identifier of the Open Forms SDK Interface Function module, as returned by openform_initialise().
- *sal_formsh* is the SAL form session handle that identifies the work item, as passed to your Open Form application by the calling script. See Creating the Script to Call the Open Forms Application.
- *lpfield_name* is the name of the iProcess field that you want to set the value of.
- *lpfield_value* is the value you want to assign to the field. **You must ensure that this value is appropriate for the iProcess field type.**

  For example, if *lpfield_name* is a iProcess date then *lpfield_value* must be a valid iProcess date string (as defined in the *SWDIR***etc\staffcfg** file.)

Description

Use this function to set the value of an iProcess field for the work item associated with the given form session.

Note:

- Most iProcess system fields (SW_*xxx*) are read only and cannot be changed using openform_set_field_value(). If you try to change a system field the call is simply ignored. No error is returned.
- It is more efficient to pass only field values that have actually changed back to iProcess. (This is because any field value that is returned by openform_set_field_value() is flagged by iProcess as changed - even if its value is the same - and passed to and from the iProcess Engine.)

Returns

One of the following numeric values.

Value	Description
1	Success.
0	Invalid *openform_id.*
-1	Invalid *sal_formsh.*
-2	Invalid *lpfield_name.*
-4	np_sal_frm_getdata interface() function not available.

openform_set_recalc()

Enable or disable recalculation of calculated fields and conditional text after an openform_set_field_value() call.

Synopsis

C/C++

```
long int openform_set_recalc (long int openform_id, long int sal_formsh, long int recalc_on)
```

VB

```
Function openform_set_recalc (openform_id As Long, sal_formsh As Long, recalc_on As Long,) As Long
```

where:

- *openform_id* is the identifier of the Open Forms SDK Interface Function module, as returned by openform_initialise().
- *sal_formsh* is the SAL form session handle that identifies the work item, as passed to your Open Form application by the calling script. See Creating the Script to Call the Open Forms Application.
- *recalc_on* should be 0 (zero) to disable recalculation, or any non-zero value to enable recalculation.

Description

By default, after an openform_set_field_value() call the SAL recalculates any calculated fields and conditional text in the iProcess Form definition for the step.

Calling openform_set_recalc():

- Switches this recalculation behavior on or off. This can provide improvements in set value performance when the iProcess Form also contains data behind your Open Forms application.
- Forces the iProcess Form window to update itself using the recalculated data if *recalc_on* is non-zero.

Returns

One of the following numeric values:

Value	Description
1	Success.
0	Invalid *openform_id.*
-1	Invalid *sal_formsh.*
-2	Unexpected error.
-4	np_sal_frm_setlong interface function() not available.

openform_terminate()

Unload the NPSAL support libraries and terminate access to the other functions in this module.

Synopsis

C/C++

```
long int openform_terminate (long int openform_id)
```

VB

```
Function openform_terminate (openform_id As Long) As Long
```

where *openform_id* is the identifier of the Open Forms SDK Interface Function module, as returned by openform_initialise().

Description

You should call this function when your Open Form application is closing - normally when your main window procedure function receives a WM_DESTROY message.

Returns

One of the following numeric values:

Value	Description
1	Success.
0	Invalid *openform_id*.

Using Caseless Forms

Caseless Forms enable a user to open a form window for any step of a procedure without starting or accessing a case. They can also be used to run an iProcess script (the initial command of the given form) if required.

A caseless form can be started either as an iProcess command line option (see below), or from **Work Queue Manager** or the **Tools** window (see Caseless Forms from Work Queue Manager).

Overview

Caseless forms have a number of uses:

- iProcess forms are quick and easy to develop and maintain. They can be used as caseless forms to provide “front ends” to a number of applications such as document image indexing, document production and database queries, etc.
- They can be used as sub-forms from a main iProcess form. This is useful for capturing “repeating” data rather than having a large form with many repeating sections.
- Scripts can be used in a caseless form to run programs on the iProcess Engine. The caseless form need never appear if you use the FORMCONTROL(3) function in the form’s initial script.

Caseless Forms Command Line Option

If iProcess Workspace (Windows) is started with the following tool in the command line, the form window of the step referred to is opened and the user can fill in fields as usual, but no case is started and there is no further action on closing the form (apart from running the Release or Keep form command if any).

```
STAFFW runstep procname stepname [datafile[delete]]
```

where:

- *procname* is the short name of the procedure, optionally followed by the host name of the procedure in the form *proc@host*.

  If the *@host* part is omitted, the server the user is logged into is assumed.

- *stepname* is the step whose form is to be opened.
- *datafile* is optional and, if included, should be the full pathname of a text file in **abox** format, containing initial values for specified fields.
- *delete* is optional, and determines which files to delete after starting the caseless form.

Value	Files to delete
0	delete no files
1	delete memo files
4	delete *datafile*
5	delete both memo files and *datafile*

  If the delete parameter is omitted, `0` (delete no files) is assumed. To include this parameter without a datafile, enter just a hyphen (–) as the latter.

If iProcess Workspace (Windows) is already running, the current instance is re-used.

Caseless Forms from Work Queue Manager

You can add one or more RunStep buttons and menu options to the user's Work Queue Manager window by editing the *SWDIR*`\etc\`*language*`.lng\staffico` file. For more information, see *TIBCO iProcess Engine Administrator's Guide*. Choosing these options open the specified forms.

Processing Caseless Forms

The form is run as if it had been opened from a work queue:

1. The Initial form command is run.
2. The **Form** window is displayed as usual, allowing user input (unless the Initial form command prevents this with the FORMCONTROL function).
3. When the form is closed, there is no Release/Keep dialog, the form simply disappears and the Release form command is run if all Required fields are filled, otherwise the Keep form command is run.

Data Accessible from the Form

Most iProcess fields are accessible from the form. Procedure defined fields will have the value SW_NA unless they have been initialized with an **abox** file. System fields have their usual values, with the following exceptions:

Field	Value
SW_GROUP:*attribute*	Attribute for the current user
SW_STARTER:*attribute*	Attribute for the current user
SW_CASEDESC	SW_NA (unless created as a procedure field)
SW_CASENUM	SW_NA
SW_CASEREF	SW_NA

Using Event Steps

You can use Event steps to control the flow of a case (such as starting a parallel branch) and to change the case data while a case is in progress.

Overview

You can use event steps to perform the following tasks:

- Suspending the Flow of a Case until an action external to iProcess takes place (for example, to wait for a response to a letter).
- Starting a Parallel Branch of a Case.
- Pausing a Case for a specific period of time (thereby providing a “diary” function).
- Externally Updating Field Data in a Case, independently of updates that result from the normal processing of work items.

The following sections describe each of these uses and describe how to create an event step and how to trigger one from iProcess.

Creating an Event Step

To create a new event step, perform the following procedure:

1. Click the **Event Step** tool.
2. Click the mouse at the required place in the SPD window.
3. Enter the Event Step **Name** (up to 8 alpha-numeric characters) and **Description** (up to 24 alpha-numeric characters) in the Properties pane.
4. (Optional) Click **Deadlines** if you want to set a deadline on the event step.
5. (Optional) Click **Duration** if you want to define a duration for the event step.
6. Click **Apply** when you have finished.

You can examine or modify an existing event step definition in the same way as editing any existing iProcess step. For more information, see *TIBCO iProcess Modeler Getting Started* or *TIBCO iProcess Modeler Basic Design*.

Triggering an Event

An event step has no addressee, and so is not delivered to a work queue. Instead, the event step is marked as outstanding by the TIBCO iProcess Engine until an event is triggered.

Events are triggered in one of following three ways:

- By expiry of the deadline (if defined), in which case the deadline action will be processed, i.e. the link from the bottom of the Event object.
- By setting procedure events, in which case events are triggered before or after performing the purge, close, resurrect, suspend, or resume actions.
- By using the *SWDIR***bin****swutil** utility, in which case the release action will be processed, i.e. the link from the right of the Event object.

  For more information about how to:

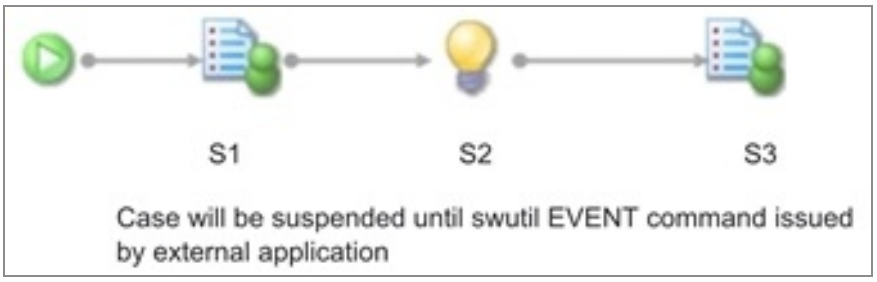
  — trigger an event from the server, see "Issue an Event" in *TIBCO iProcess swutil and swbatch Reference Guide*.

  — trigger an event from iProcess Workspace (Windows), see "Issue an Event" in *TIBCO iProcess Workspace (Windows) Manager's Guide*.

Note: Entries are put in the Audit Trail of the form “... processed to [EVENT]” and “event issued by...” for procedures using events.

Suspending the Flow of a Case

A case can be suspended by linking to the left of an Event from the release action (i.e. the right hand side) of a Step. When the Step is released this branch of the case is suspended, and an entry is written to the Audit Trail to show that an event is outstanding.

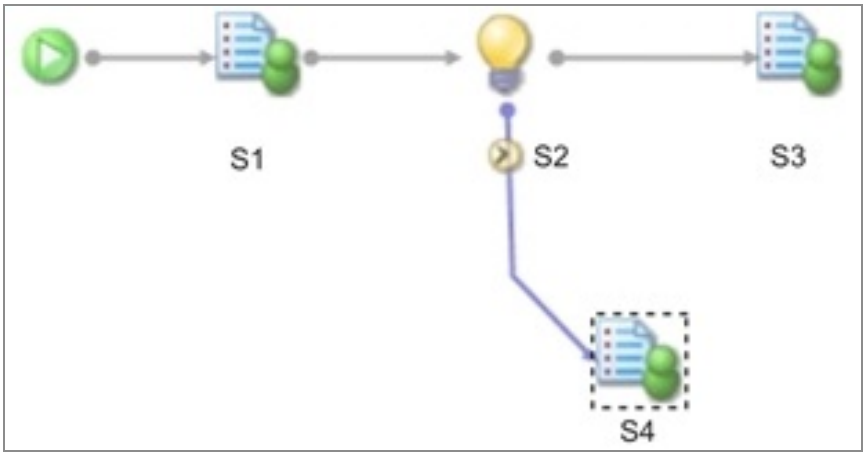


To restart the case, an external process must send a trigger to iProcess, causing the 'release' actions defined on the event step to be processed.

The trigger is sent by calling the *SWDIR***bin****swutil** utility. If required, fields may be updated by including an **abox** file in the event command line - for more information, see Externally Updating Field Data in a Case.

Note: In the example shown above, if the Event is never triggered, that branch of the case will be suspended indefinitely.

To avoid this scenario, consider the alternative shown in the following diagram. In this scenario, if the Event (**S2**) is not triggered by the external program within the defined deadline period, the case will proceed to the **S4** step (a reminder, escalation step) when the deadline expires.

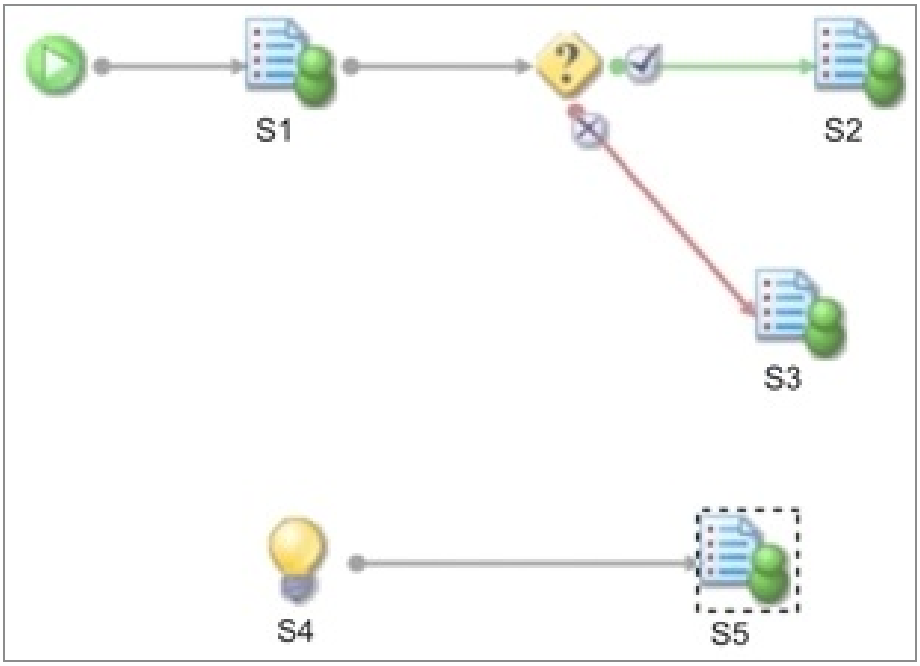


Starting a Parallel Branch of a Case

Events can be used to start up an entire branch of a case as a result of some external trigger. To achieve this effect, the Event object in the TIBCO iProcess Modeler has no links to the left hand side (and therefore no 'suspend' functionality). It does however have one

or more links from the right hand side, denoting that one or more actions will be processed when the Event is triggered.

Note: New case data may also be introduced when the Event is triggered, using an **abox** file - for more information, see Externally Updating Field Data in a Case.



For example, consider a holiday booking process. Details of a customer's requirements are initially input in an external holiday booking system and a case of an iProcess procedure is started automatically to process the necessary fulfillment activities. After the case has started, additional requirements may be input to the external system. This results in the automatic triggering of an Event, which causes the delivery of another work item which advises the caseworker of the change in requirements.

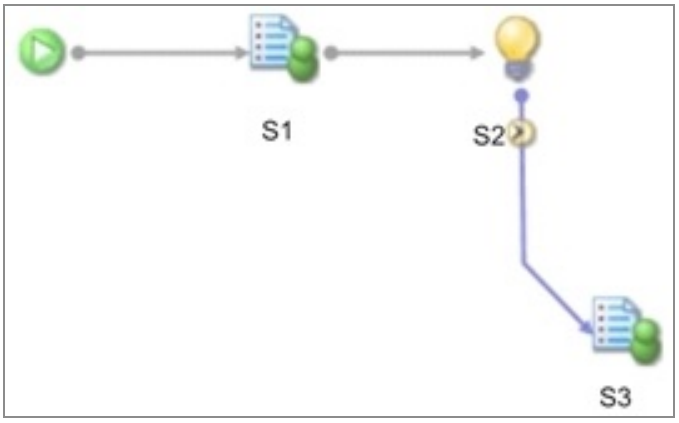
Pausing a Case

To pause a case, or a branch of a case, until a specified date and time:

1. Define an event step with a deadline in the flow of the procedure - see Creating an Event Step.

2. When the procedure reaches the event, the case is suspended until the deadline that is defined on the Event expires. The deadline actions are then issued and the case continues.

Note: The event step must have Withdraw set on the deadline, or the case never gets completed. On the **Deadlines** tab, select the **Withdraw form from queue on expiry** check box.



Externally Updating Field Data in a Case

An external process can interact with an iProcess procedure by updating field values in a case that is already running. For example, you might want to update a field in a case with a new interest rate or update the values of several CDQP parameters.

Note: This functionality is sometimes referred to as Process Variables in this guide.

To update your case data, you have to perform the following steps:

1. Define an EVENT step in your procedure.
2. Use the `swutil EVENT -p` command with an **abox** file that contains the new field values. For more information about using **swutil EVENT**, see “Events” in *TIBCO iProcess swutil and swbatch Reference Guide*.

There are a number of ways in which you can control how and where a field is updated. For example, you can change the value of a field in a sub-case without changing the value

of the field in the parent case or you can specify a new value that will be passed to a specific case started from a dynamic sub-procedure call.

The following sections describe how case data is updated and the different ways you can control how the case data is updated:

- How is the Case Data Updated with New Field Values?
- To Define an EVENT Step for Updating Case Data
- Examples of Updating Field Values
- Explicitly Updating Fields in Sub-Procedures

How is the Case Data Updated with New Field Values?

For every case of a procedure, work items for each addressee are given a working copy of the values of all assigned fields in a case from the central case data (stored in the **case_data** database table). This working copy of data is known as pack data (stored in the **pack_data** database table). The pack data is modified by users or programs working on the work item(s) so if any fields are changed, they are changed in the pack data first and then copied to the case data when the work item is released.

When updating case data, only the field values specified in the **abox** file are updated in the case. All other field values in the case are unchanged. This allows you to use **swutil EVENT** to selectively update field values in work items. (By comparison, a **swutil RESEND** completely recreates a work item, overwriting all local **pack_data** in the work item. The **RESEND** command updates all fields in all outstanding steps to the values in the case data so any changes made in steps that are Kept (not Released) are lost.)

By using different formats for the new field values in the **abox** file, you can have more control over which fields you update. For example, you can update a specific field in a particular sub-case or change the input parameter to a sub-case - see Explicitly Updating Fields in Sub-Procedures.

Note: Delayed release EAI steps will not have data updated, as there is no mechanism to transfer the updated data to the external system.

Propagation of New Field Values

This section describes how new field values supplied in an **abox** file using **swutil EVENT -p** are propagated through a case.

By default, if a new field value is supplied in an **abox** file, the new value is propagated through the parent case and down to any sub-cases via the appropriate field mappings defined by the step that initiated the sub-case. The **case_data** and **pack_data** database tables are updated for both the parent case and all affected sub-cases.

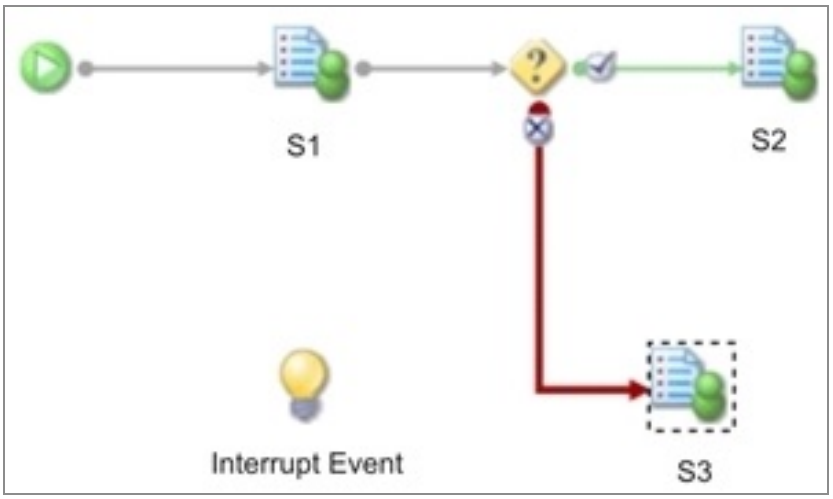
However, there are a few exceptions to this:

- If the input parameter is set from an expression or script and a field value in the expression or script is being updated, the new value is not passed to the sub-case. This is because the expression or script is not re-evaluated. You can overcome this by explicitly changing field values in the sub-case - see Explicitly Updating Fields in Sub-Procedures.
- Because Graft steps do not have input parameters, there is no automatic propagation of changed field values down to the sub-cases of a Graft step. However, you can modify sub-case fields in the same way you explicitly update fields in sub-procedures - see Explicitly Updating Fields in Sub-Procedures.

  iProcess assigns an index automatically to each sub-case that is grafted to the Graft step. This means you need to manually check that the correct index is used when supplying new case data - for example, by using the **getSubArrayIdx** function in TIBCO iProcess Objects. For more information, see *TIBCO iProcess Objects Programmer's Guide*.
- If you use Dynamic sub-procedure calls, new field values are propagated to the sub-cases because the array element index is used to pass the correct field values to the appropriate sub-case(s). However, if a single instance field is passed, the field value is passed to all the sub-cases.

To Define an EVENT Step for Updating Case Data

To update case data in your procedure, you need to define an Event step in the procedure with no actions (i.e. no links from the right or the bottom of the Event object). This Event is not part of the procedural flow, i.e. it is not an action from any other step.



An external process can then issue a **swutil EVENT -p** command, specifying the name of the event step and an **abox** file containing the new field values. The field values supplied in the **abox** file are applied to the fields for this case of the procedure.

For more information about the syntax for using *SWDIR***bin****swutil**, see "`swutil EVENT`" in *TIBCO iProcess swutil and swbatch Reference Guide*.

For more information about the format of an **abox** file, see abox Files. There are various ways you can control what fields are updated in the case by specifying a different field format in the **abox** file.

Examples of Updating Field Values

The following examples demonstrate how you can use *SWDIR***bin****swutil** with an **abox** file to update a field in a main procedure (Example 1) and in a sub-procedure (Example 2):

Example 1

For the procedure called CARPOOL, case number **100**, event step called EVENTUPDATE where you want to modify the value in the INTRATE field from **5** to **4.5**, enter the following:

```
swutil EVENT carpool 100 eventupdate c:\temp\abox1 -p
```

where the **abox1** file contains the following entry:

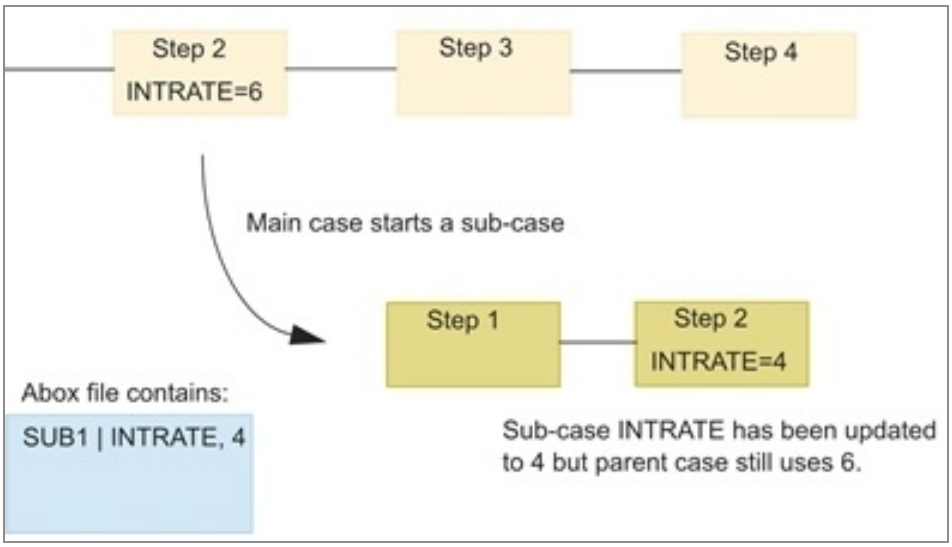
```
INTRATE, 4.5
```

The new value of **4.5** is automatically propagated down to any sub-cases in case number **100** that use this field.

Example 2

If you want to change a specific field value called INTRATE in a sub-case with a sub-procedure call step name of SUB1 without changing the calling field value (in the parent case), enter the following in the **abox** file:

```
SUB1|INTRATE, 4
```



For more information about the format of **abox** files, see abox Files and for the complete command line syntax for **swutil EVENT**, see *TIBCO iProcess swutil and swbatch Reference Guide*.

Explicitly Updating Fields in Sub-Procedures

There are three ways in which you can update a sub-procedure field using Process Variables. Each method is listed below in the order of precedence (highest first).

- Explicitly updating a sub-case field.
- Explicitly updating a sub-case input parameter.
- Automatically propagating parent field assignments - see Propagation of New Field Values.

This means that if a new field value is updated in a parent case and also explicitly in a sub-case, the new value defined for the sub-case takes precedence over the automatically propagated value.

Because of the different ways sub-procedures can be called there are slight differences in how updated fields are propagated to the sub-cases or how you explicitly change a sub-case field. The following table shows the sub-procedure call types and which methods can be used to update fields in the sub-cases:.

Sub-Procedure Type	Automatically Propagate Parent Fields to Sub-Cases	Explicitly Update a Sub-Case Field	Explicitly Update a Sub-Case Input Parameter
Static	Yes	Yes	Yes
Dynamic	Yes	Yes	Yes
Graft	No	Yes	No

The following sections describe specific information related to the way process variables work with each sub-procedure type.

Static Sub-Procedures

- Any fields that you update in the parent case are automatically propagated to the sub-case.
- You can explicitly change a field in the sub-case.
- You can explicitly change a sub-case input parameter without changing the field value in the parent case. This can only be done when I/O parameters have been defined for the sub-procedure. You map the new field value to the appropriate $IPT*n* value.
- If a field for a sub-case is explicitly specified in an **abox** file and a sub-case is not already outstanding for that step, the sub-case is started automatically.

Dynamic Sub-Procedures

- Array fields are used to pass values from the main procedure to the required sub-procedures. You can change one element in an array field and this is propagated to

the relevant sub-case (using the array element index).

- If a single instance field is passed as a parameter and is changed via Process Variables, it is propagated to all sub-cases started from that step.
- Any fields that you update in the parent case are automatically propagated to the sub-case.
- You can explicitly change fields in the sub-cases.
- You can explicitly change a sub-case input parameter. This can only be done when I/O parameters have been defined for the sub-procedure. You map the new field value to the appropriate $IPT*n* value.
- When explicitly changing a value in a sub-case and no outstanding sub-case for that index exists, the sub-case is started automatically and the new value is propagated down to it.

Graft Steps

- There is no automatic propagation of parent case fields to Graft steps because there are no input parameters to Graft steps.
- You can explicitly change fields in the sub-cases.

For more information about the syntax you need to use in the **abox** file to update case data in these sub-procedure types, see abox Files.

Caveats When Updating Field Values

There are a number of important points to note when using Process Variables:

- If a field value that is used as a CDQP parameter is specified in the **abox** file, the new value is used to display, sort and filter in the Work Queue Manager the next time the work queue is refreshed.
- If the user has changed a field specified in the **abox** file and then kept the work item, that change is lost. The original modified value is overwritten by the value specified in the **abox** file.
- If a work item is open when the event updates its **pack_data**, when the user tries to keep or release the work item they will receive an error message informing them that the work item has been updated elsewhere since they opened the work item. The keep or release operation is canceled and any changes to field values made by

the user are lost. Updates to field values made by the event are applied. (The user can then re-open the work item and make their changes again.)

Configuring Activity Monitoring

You can configure the TIBCO iProcess Engine to publish iProcess Engine activity information to external applications.

Overview

The TIBCO iProcess Engine can be enabled to publish iProcess Engine activity information to external applications. An activity is any instruction in the iProcess Engine that creates an audit trail entry, for example, **Case started** or **Event Issued**. You can configure any combination of step and/or activity to be monitored. This enables an external application to monitor important business events during the processing of cases.

A **BG** process can identify if a step is being processed and if activity monitoring has been configured for it. The **BG** process then sends details of the configured activities in XML format to the **IAPJMS** (Introspection Activity Publication JMS) process.

The **IAPJMS** process sends the XML message to a specified JMS topic, from which an external application (for example, TIBCO iProcess Objects, iProcess Analytics, or an external application that you have written) can receive the messages.

For information on how to administer activity monitoring, see "Administering Activity Monitoring" in *TIBCO iProcess Engine Administrator's Guide*.

Auditable Objects and Activities

Activity monitoring enables you to monitor the detailed transactions for an individual case of a procedure. For example, you can monitor when steps are processed, when deadlines are reached, the output from fields at a particular point in the procedure and when the case has started and finished.

You can monitor any of the activities that are reported in the audit trails when cases are processed. Audit trails are a detailed log of all transactions for an individual case of a procedure.

For an explanation of the activities that you can monitor, see "Understanding Audit Trails" in *TIBCO iProcess Engine Administrator's Guide*.

How to Configure Activity Monitoring Information

To configure activity monitoring information for your iProcess Engine, you must perform the following steps:

1. Generate your activity monitoring configuration information as XML in the form of a Message Event Request (MER) message. See Understanding Message Event Request (MER) Messages. You can do this using any XML tool, for example, Altova XMLSpy. The MER message should conform to the `SWMonitorList.xsd` schema. See Understanding the Activity Monitoring Schemas.
2. Once you have generated a MER message according to your requirements, there are two ways to update or replace the activity monitoring configuration information in the iProcess database with the activity monitoring configuration information in the MER message:
   - using TIBCO iProcess® Server Objects, see *TIBCO iProcess Server Objects Programmer's Guide*.
   - by using the *SWDIR*`\bin\swutil IMPMONITOR` and `EXPMONITOR` commands, see "Monitoring Activities" in *TIBCO iProcess swutil and swbatch Reference Guide*.

Understanding the Activity Monitoring Schemas

When you install the TIBCO iProcess Engine, three schemas are installed which are used to configure the activity monitoring configuration information. The schemas are installed in the *SWDIR*`\schemas` directory. The schemas are:

- `SWMonitorList.xsd`. The format of the Monitor Event Request (MER) messages must conform to this schema. MER messages request the activities to monitor and are sent to the iProcess database to update the activity monitoring configuration information. See How to Configure Activity Monitoring Information.
- `SWAuditMessage.xsd`. The format of the Monitor Event Detail (MED) messages must conform to this schema. These messages can be sent in either a basic format or an

extended format conveying additional information. MED messages contain the information about the activities being monitored and are sent from the **IAPJMS** process to the external application. For more information, see "Administering Activity Monitoring" in *TIBCO iProcess Engine Administrator's Guide*.

- `SWType.xsd`. This defines the iProcess field types that are used in the `SWMonitorList.xsd` and the `SWAuditMessage.xsd` schemas.

Understanding Message Event Request (MER) Messages

Every MER message sent to the iProcess database to update the activity monitoring configuration information consists of XML requesting the events to monitor. The MER XML format is defined by the `SWMonitorList.xsd` schema.

Note that:

- You can use an activity ID of **-1**. Using an activity ID of **-1** is the equivalent of using `$ALL$` for a step name. It means all activities are monitored. For example:

  ```
  <Activity Num="-1"/>
  ```

  For an explanation of the activities that you can monitor, see "Understanding Audit Trails" in *TIBCO iProcess Engine Administrator's Guide*.

- For the following activity IDs:

  000, **007**, **008**, **009**, **021**, **022**, **023**, **024**, **057**

  You must set the value of `<Step Name>` to `$ALL$`. For example:

  ```
  <Step Name="$All$">
  ```

  If you do not, then no MED message is generated. This is because the schema enforces that a value for the **step name** element must be defined for these activities. However, because of the way the iProcess Engine works, there is no step name for these activities. For example, for activity ID **000**, there is no step name when a case is started because the case has not yet reached its first step. Therefore, to work around this, you should specify a value of `$ALL$` in the **step name** element for these activity IDs.

Examples of Configuring Activity Monitoring Information

The format of the Monitor Event Request (MER) message must conform to the `SWMonitorList.xsd` schema. When generating a MER message, you can specify the following:

- the name of the procedure you want to monitor.
- names of the fields whose data you want to output. You can specify all fields for all activities or you can specify a particular field for a particular activity, depending on your requirements.
- activities you want to monitor or you may specify that you want to monitor all activities. When specifying the activities that you want to monitor you must also specify the steps on which the activity will be monitored. However, you may specify that you want to monitor an activity on all steps in that procedure. For an explanation of the activities that you can monitor, see "Understanding Audit Trails" in *TIBCO iProcess Engine Administrator's Guide*.

Example

This section describes an example of:

- a MER message generated to configure activity monitoring configuration.
- an MED message that is generated as a result of the example configuration.

The MER message detailed below shows activity monitoring information for a procedure called CARPOOL. The table below describes:

- the activities that are to be monitored.
- the steps that those activities are to be monitored on.
- the field data to be published when the activity occurs.

Activity	Step Name	Field Data
Case Start (0)	ALL	
Work item Released (2)	REQUEST	PURPOSE

The MER message generated to represent the information in the table above is shown below.

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<ProcedureMonitor xmlns="http://bpm.tibco.com/2004/IAP/1.0/MER"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://bpm.tibco.com/2004/IAP/1.0/MER
SWMonitorList.xsd">
<SchemaVersion>1.0</SchemaVersion>
<MessageType>MER</MessageType>
<FullImport>false</FullImport>
<MonitorDetail>
<Procedure Name="CARPOOL">
<NodeName>swnod103</NodeName>
</Procedure>
<GlobalFieldList>
<Field Name="STARTDATE"/>
<Field Name="VEHICLE"/>
</GlobalFieldList>
<MonitorList>
<MonitorLine>
<ActivityList>
<Activity Num="0"/>
</ActivityList>
<StepList>
<Step Name="$All$"/>
</StepList>
</MonitorLine>
<MonitorLine>
<ActivityList>
<Activity Num="2"/>
</ActivityList>
<StepList>
<Step Name="REQUEST"/>
</StepList>
<FieldList>
<Field Name="PURPOSE"></Field>
</FieldList>
</MonitorLine>
</MonitorList>
</MonitorDetail>
</ProcedureMonitor>
```

The XML below demonstrates some output MED messages that have been generated from the MER message.

```
<MED|CARPOOL|vora677|uk-nickh2k3-ora>
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<AuditEvent xmlns="http://bpm.tibco.com/2004/IAP/1.0/MED"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://bpm.tibco.com/2004/IAP/1.0/MED
SWAuditMessage.xsd">
<SchemaVersion>1.0</SchemaVersion>
<MessageType>MED</MessageType>
<ActivityID>0</ActivityID>
<Procedure Name="CARPOOL">
<NodeName>vora677</NodeName>
<Number>3</Number>
<Description>Company Car Allocation</Description>
<MajorVersion>1</MajorVersion>
<MinorVersion>0</MinorVersion>
<Type>MAIN</Type>
<Status>model</Status>
</Procedure>
<Case Number="72">
<Description>Case 1</Description>
<Starter>swadmin@vora677</Starter>
<TimeStarted Microseconds="674419">2005-07-26T15:59:22</TimeStarted>
</Case>
<AuditMessage>Case started by swadmin@vora677</AuditMessage>
<Field Name="STARTDATE">
<Type>DATE</Type>
<Value></Value>
</Field>
<Field Name="VEHICLE">
<Type>TEXT</Type>
<Value></Value>
</Field>
</AuditEvent>
<MED|CARPOOL|vora677|uk-nickh2k3-ora>
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<AuditEvent xmlns="http://bpm.tibco.com/2004/IAP/1.0/MED"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://bpm.tibco.com/2004/IAP/1.0/MED
SWAuditMessage.xsd">
<SchemaVersion>1.0</SchemaVersion>
<MessageType>MED</MessageType>
<ActivityID>2</ActivityID>
<Procedure Name="CARPOOL">
<NodeName>vora677</NodeName>
<Number>3</Number>
<Description>Company Car Allocation</Description>
<MajorVersion>1</MajorVersion>
```

```
<MinorVersion>0</MinorVersion>
<Type>MAIN</Type>
<Status>model</Status>
</Procedure>
<Case Number="72">
<Description>Case 1</Description>
<Starter>swadmin@vora677</Starter>
<TimeStarted Microseconds="674419">2005-07-26T15:59:22</TimeStarted>
</Case>
<AuditMessage>"REQUEST" released by swadmin@vora677</AuditMessage>
<AuditStep Name="REQUEST">
<Description>Request for Vehicle</Description>
<AuditDate Microseconds="429174">2005-07-26T15:59:54</AuditDate>
<AuditUser name="swadmin@vora677">
                <Description>Admin user</Description>
                <Type>User</Type>
</AuditUser>
<Addressee Name="swadmin@vora677">
                <Description>Admin user</Description>
                <Type>User</Type>
       </Addressee>
</AuditStep>
<Field Name="STARTDATE">
<Type>DATE</Type>
<Value>27/06/2005</Value>
</Field>
<Field Name="VEHICLE">
<Type>TEXT</Type>
<Value>Estate</Value>
</Field>
<Field Name="PURPOSE">
<Type>TEXT</Type>
<Value>House moving</Value>
</Field>
</AuditEvent>
```

Using EIS Reports

You can create reports on iProcess case data and then view them using a third party data viewer application (or any program that can read **csv** files). iProcess provides the Executive Information System (EIS) components so that you can create a case data report, which you can view in a third-party EIS viewer.

Note: For a full description of how to use the EIS components to produce reports on iProcess data, see "Reporting on Case Data" in *TIBCO iProcess Workspace (Windows) Manager's Guide*.

iProcess EIS Components

- iProcess Case Data Extraction utility *SWDIR***util****swcdata.**
- EIS Report object(s) in a procedure definition.
- EIS Report tool(s) in a user's **Tools** window or Work Queue Manager.
- Third-party EIS Data Viewer (editor).
- Third-party EIS Data Viewer (runtime).

Defining an iProcess EIS Report

To define an EIS report, perform the following steps:

1. Run some test cases on the procedure.
2. Edit the procedure with the Process Definer and create an EIS Report object which determines the **csv** file format and contents.

   Note: The following steps may be done from within the **EIS Report Definition** pane.

a. Create a test **csv** file (on the **Data File** tab, use the **Create Test File** button).

b. On the **Data Viewer** tab, under **Data Viewer Project Editor**, click **Run Editor** to run the EIS Data Viewer and create the report format.

c. Under **Run-Time Data Viewer**, click **Run Viewer** to run the EIS Data Viewer and test the report.

d. Click the **Access** tab to restrict runtime access to the EIS Report to particular users, roles, or expressions.

3. Click **Apply** to save the procedure.

   A procedure may have any number of EIS Report objects, producing different reports.

4. Ensure that the *SWDIR***etc***language***.lng\staffico** file on the server contains the relevant entries (SWEIS).

The EIS Case Data Extraction Utility

This utility (*SWDIR***util****swcdata**) runs on the server to extract case data and then puts it into a **csv** file, which can be viewed by any data viewer capable of reading **csv** files. The data to be extracted, and the format to be output, may be defined in one of two ways:

- An EIS Report object created in the **Process Definer** window. This is the easiest way, and the report can then be run from the iProcess Workspace (Windows) Work Queue Manager window.
- An ASCII command file created on the Server. This enables a system integrator to specify the extraction criteria without requiring access to the Process Definer. The **csv** file can then be picked up by the data viewer to display the report.

Note: The extraction utility can be run automatically as a batch job, for example overnight, or manually. Also single cases or ranges of cases can be appended to an existing **csv** file: see below.

Only iProcess users with Case Administration access permission for the procedure (see "Controlling Access to Procedures" in *TIBCO iProcess Modeler Procedure Management*), and the Procedure Owner and the System Administrator, can run this utility.

EIS Command File

This is a file used to specify the case data to be extracted, and the format to be output to a **csv** file for display by the EIS Data Viewer.

It is used with the *SWDIR***util****swcdata** utility as an alternative to defining an EIS Report object.

It is an ASCII file consisting of lines of commands each consisting of a keyword separated by white space from parameters as shown below. Most commands are optional and default as indicated. Keywords are case insensitive and the commands may be in any order in the file.

Command	Description / Valid Values
TYPE CSV	File output type: **csv** is the only currently supported format and is the default.
CASETYPE	**ALL**, **TERMINATED** or **ACTIVE**.
COLUMNS	*This command is obligatory* and should be followed by a newline and then a list of the field names to be extracted, one per line and terminated by a line consisting of a single tilde character, ~. Any of the user-defined fields may be used plus certain special fields - see Special Fields for EIS Report Columns.
FIELDSEP	The string of characters to be used to separate each field item on a line (default is a comma). Leading white space is ignored (and the SPACE character is not allowed). The following special character sequences may be used: **\r** carriage return **\n** newline **\t** tab **\f** form feed \\ backslash
RECORDSEP	The string of characters to be used to separate each line of field items (corresponding to a Case) - default is **\n** (newline). Leading white space is ignored (and the SPACE character is not allowed). The same special

Command	Description / Valid Values
	character sequences may be used as for FIELDSEP above.
QUOTE	The character to be placed at each end of non-numeric fields. The default is a double quote character (").
HEADER	**YES** or **NO**: specifies whether to include the names of the fields to be extracted (as quoted strings) as the first line of the file (default is **YES**).
SERVERFILE	The simple filename of the output **csv** file. **This command is obligatory**. The file is placed in the *SWDIR***tsys\reports***host***.n** directory where *host* is the procedure hostnode.
OWNER	The required owner of the **csv** file. (Default - the user who is running the utility.)
PERMISSIONS	File permissions of the **csv** file according to the UNIX convention, rwxrwxrwx with a: • **-** where a parameter is missing • **r** = readable • **w** = writable The first 3 characters refer to the owner, the second 3 the group and the last 3 all users. (The default is the user default file permissions.)

Special Fields for EIS Report Columns

The following special fields can be used in EIS columns:

Field	Description
SW_CASEDESC	Case Description
SW_CASENUM	Case Number (numeric)

Field	Description
SW_CASEREF	Case Reference in the form *pp-nn*
SW_DATE	Date of data extraction: DD/MM/YYYY
SW_DATETIME	Date and time of data extraction, separated by a space: DD/MM/YYYY HH:MM
SW_PRODESC	Procedure Description
SW_PRONAME	Procedure Name
SW_STARTER	Username of the case starter
SW_STEPDESC	EIS Report object description
SW_STEPNAME	EIS Report object name
SW_TIME	Time of data extraction: HH:MM

iProcess Commands

This section describes how to use iProcess commands to add extra functionality to your procedures.

Overview

An iProcess command is a way of associating a particular operation with either a form or a field. Each command can be any iProcess expression, but usually it will be one of the following:

- A function call to run an **external program**, such as SERVEREXEC (run a server program), or WINRUN (run a program on a graphical client). For more information, see *TIBCO iProcess Expressions and Functions Reference Guide*.
- A call to an iProcess **script**, which is a collection of expressions, conditions and loop constructs. Use the **Call** (*script_name*) function to call a script - for information about how to create a script, see "Using Scripts" in *TIBCO iProcess Modeler Advanced Design*.
- An **assignment** expression to give a new value to a field, for example:

  ```
  FIELD2 := SUBSTR (FIELD1, 1, 2)
  ```

  This would assign part of FIELD1 to FIELD2 when the command is run. For more information about the allowable assignment expressions, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Typical uses might be:

- Interactive database lookup, providing data for other fields in the procedure.
- Opening an associated document image window.
- Retrieving data from another application such as a spreadsheet.
- Providing the user with external help in completing the field.

Form Commands

There are three kinds of form command:

Command	Description
Initial	This command is run when the work item form is opened from the user's Work Queue.
Keep	This command is run when the form is returned to the user's Work Queue.
Release	This command is run when the form is released.

Form commands are defined in the Properties pane - for more information about entering the commands, see *TIBCO iProcess Modeler Basic Design*.

Field Commands

Field commands are run when a field is opened, by using one of the following options:

- Clicking the field's **Open** button.
- If the field is defined as Auto-Open, the field command is run automatically on pressing ENTER, or moving from the field after changing its value.

For more information about how to define commands on fields, see *TIBCO iProcess Modeler Basic Design*.

External Validation Lists

This is a facility that enables you to provide a validation list for a field externally to iProcess.

You do this by entering a special function (with appropriate parameters) in the field marking validations **Values** column (of a required or optional field). There are two functions available:

Function	Description
VLDFILE	Adds lines from a text file to the list of validations of the current field.
VLDQUERY	Adds values from the integrated database (for example, Oracle) to the list of validations of the current field.

Note: VLDQUERY is only applicable if the iProcess Engine is integrated with a database.

For more information about these functions, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Controlling Windows

The following functions may be useful for controlling windows belonging to iProcess (or other applications). For more information about these functions, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Function	Description
ISWINDOWS	Check if running a 16-bit Windows Client
FORMCONTROL	Perform action on current form
WINEXIST	Check if a window exists
WINFIND	Find a window to perform an action on
WINACTION	Perform an action (such as close, move, resize) on a window
SENDKEYS	Send keystrokes to the active window
WINMESSAGE	Display comfort message in window

Function	Description
MESSAGEBOX	Display message box
FILEREQUEST	Request file selection

Running External Programs using iProcess Functions

The following functions can be used to run external programs (on the server or client) from iProcess. For more information about the functions, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Function	Description
SERVERRUN	Run a server program
SERVEREXEC	Run a server program (no shell)
WINRUN	Start a program on a graphical client

abox Files

These are ASCII text files that certain integration options use to pass data to iProcess. The data is used to update field values in the current case of the procedure - for more information, see Externally Updating Field Data in a Case.

To change the value of a field in a main procedure, the file should contain lines consisting of the name of the field, followed by a comma, followed by the data. For example:

```
CUST_BALANCE,400.10
```

However, the **abox** file format enables you to be more precise in what field you update - for example, you might want to update a field value in a sub-case without changing the

field value in the parent case or update a field within a composite field. The following table describes the possible formats you can use in the **abox** file and what is updated:

Abox Syntax	Description
FIELDNAME, *new_value*	Sets *FIELDNAME* to the value supplied as *new_value.*
FIELDNAME :*FIELD1*, *new_value*	Sets *FIELD1* in the composite field of *FIELDNAME* to the value supplied as *new_value.*
SUBCALL | *FIELD*, *new_value*	Sets the value of *FIELD* in the sub-procedure called from the SUBCALL step name to the value supplied as *new_value.*
SUBCALL | $IP*n*, *new_value*	Sets the value of the field mapped to the $IP*n parameter* in the sub-procedure called from the *SUBCALL step* to the value supplied as *new_value*. **Note:** You can get the name for a parameter ($IP*n*) by looking at the sub-procedure Input mapping dialog box.
DYNCALL [*index*] | $IPT*n*, *new_value*	Sets the field mapped to the $IPT*n* parameter in a sub-case started from a dynamic sub-procedure call to the value supplied in *new_value*. The index is used to determine which sub-case is affected e.g. if *index* is 10 the tenth sub-case started would have its field updated. **Note:** You can get the name for the parameter ($IP*Tn)* by looking at the dynamic sub-procedure Input mapping dialog box.
DYNCALL [*index*] | *FIELD*, *new_value*	Sets *FIELD* in a sub-case started from a the dynamic sub-procedure call step name of *DYNCALL* to the value supplied in *new_value*. The index is used to determine which sub-case to update.

For example, the following are valid entries in an **abox** file:

```
CUST_NAME,Algernon Fitzpatrick
CUST_BALANCE,400.10
```

```
SUB1|CUST:POSTCODE,SN1 6EN
SUB1|SUB2|ACCOUNT_NAME,Jefferson
DYNCALL[10]|NAME, Paul
SUBCALL|$IP1, Patrick
```

If there is no text after the comma, the field is set to SW_NA (unless it is a TEXT field, in which case it is set to blank but assigned). Fields can also be set to SW_NA by putting the text “SW_NA” after the comma - for example:

```
FIELD,SW_NA
```

Note:

- Text data is NOT enclosed in quote marks
- Delimiters are NOT used for date or time data – just enter the figures with the appropriate separators, e.g. DD/MM/YYYY for dates and HH:MM for times.
- Dates should always be provided with a full 4 digit year specification.
- For a **memo** field, the value is the pathname of a text file containing the memo text.

Scripts

A script is a collection of statements that can be called from various places within iProcess (for example, when a field is opened). Scripts are useful when more than one iProcess expression is needed to achieve a command’s requirements.

To define a script:

1. Create the script as described in “Using Scripts” in the *TIBCO iProcess Modeler - Advanced Design* guide.
2. Define the script form using the Script Editor.

To call a script:

Define a form or field Command consisting of the function CALL ("*script*") where *script* is the name of the script. You can also use the SCRIPT function to call a script.

For more information about using the CALL or SCRIPT functions, see *TIBCO iProcess Expressions and Functions Reference Guide*.

Transforming iProcess Procedures into the Workflow Standard XML Process Definition Language (XPDL)

This section describes how to transform iProcess procedures into the Workflow Standard XML Process Definition Language (XPDL) and vice versa. XPDL is developed by the Workflow Management Coalition (WfMC).

Overview of the WfMC and XPDL

This section is an overview of the Workflow Management Coalition (WfMC) and the Workflow Standard XML Process Definition Language (XPDL).:

- About the WfMC
- About XPDL
- Why Use XPDL?

About the WfMC

The Workflow Management Coalition (WfMC) is a non-profit making organization whose members are workflow vendors, users, analysts and university/research groups. They describe their mission as ‘to promote and develop the use of workflow through the establishment of standards for software terminology, inter-operability and connectivity between workflow products’. For more information about the WfMC, see www.wfmc.org.

About XPDL

The WfMC has developed the Workflow Standard XML Process Definition Language (XPDL). The WfMC explain that “XPDL provides a framework for implementing business process management and workflow engines, and for designing, analyzing, and exchanging

business processes". For more information about the XPDL standard, see www.wfmc.org. Information about XPDL in this guide has been referenced from www.wfmc.org/standards/docs/TC-1025_10_xpdl_10250.pdf.

Why Use XPDL?

The benefit of XPDL is that it enables inter-operability between workflow systems. This is achieved by describing business processes in the smallest amount of entities possible, so that it enables process definitions from other vendors to be easily transformed into XPDL. This means that, as long as the vendor supports XPDL, process definitions can be used by different vendors, regardless of the workflow system that was used to produce them originally. For example, you can take a process definition that has been created by third party vendor, Enhydra™ JaWE (Java Workflow Editor) and load it into iProcess and vice versa.

Understanding XPDL Translation Scenarios

The ability to transform iProcess procedures into XPDL introduces two transformation scenarios:

- You can transform an iProcess procedure into XPDL.
- You can transform some XPDL source into an iProcess procedure. The XPDL source can be:
  - created manually.
  - created from another vendor.

Understanding Translation Concepts

Some iProcess procedure entities map to XPDL entities and some do not. Therefore, you need to understand how iProcess procedures are transformed into XPDL and vice versa. This is especially true if you are going to perform the transformation more than once. For example, transforming an iProcess procedure to XPDL and then back to iProcess. This is because you may lose something that you have configured in the iProcess procedure because it is not recognized in XPDL.

It contains the following sections:

Overview

When transforming XPDL into an iProcess procedure, if the XPDL source uses concepts or entities that are not supported by iProcess, the concept is either ignored or converted to the nearest possible concept within iProcess. This means that usually an incomplete procedure is created. The procedure can be edited in the TIBCO iProcess Modeler to make it complete.

When transforming an iProcess procedure into XPDL, where the iProcess procedure and XPDL entities map, the iProcess procedure entities are converted to XPDL entities. The parts of the iProcess procedure that do not map are stored as extended attributes.

All iProcess procedure XPDL extended attributes are optional. This means that iProcess Workspace (Windows) will load XPDL that does not have any iProcess extended attributes defined. See Understanding XPDL Produced From an iProcess Procedure.

How iProcess Procedure Entities and XPDL Entities Map Together

To understand how XPDL transforms into an iProcess procedure and vice versa, you need to understand how iProcess entities map to their equivalent XPDL entities. The following table gives a general description of how iProcess entities map to their equivalent XPDL entities. For more detailed information about how iProcess entities transform into XPDL entities, see How iProcess Transforms to XPDL.

iProcess	XPDL
Procedure. A procedure consists of	Workflow Process. A Workflow Process consists of

iProcess	XPDL
steps. Each step is a unit of work. The procedure defines the relationships between those steps.	activities. Each activity is a unit of work. The workflow process defines the relationships between those activities.
Fields. Fields store the business data that is created and used in the procedure.	Workflow Relevant Data. The business data that is created and used as part of the workflow process.
iProcess Users and Groups. The user/group on whose behalf the actions are performed, as a result of the step.	Participants are descriptions of resources that can act as the performer of the various activities in the process definition.
Form Definitions and EAI steps. The form definition is displayed at run-time and enables the user to amend information required by the process. EAI steps enable integration between external applications and iProcess.	Applications. The applications or interfaces which may be invoked by the workflow service to support, or wholly automate, the processing associated with each activity.
Normal Steps, EAI Steps, Sub-Procedure Call Steps, Dynamic Sub-Procedure Call Steps, Graft Steps. Each of these steps can be used to perform a specific activity that make up the workflow process.	Activity. The activities define each elementary activity that makes up a workflow process.
Links. Data in the workflow process is processed depending on how the steps are linked together.	Transitions. Activities are linked by transitions. Each individual transition has three elementary properties, the from-activity, the to-activity and the condition under which the transition is made. The transitions within a process may result in the sequential or parallel operation of individual activities within the process.

XPDL Entities That do Not Map to iProcess Procedure Entities

This section describes the entities in iProcess procedures and XPDL that do not map to each other.:

Activity Sets

If you have any activity sets in your XPDL source, they are ignored by iProcess when the XPDL source is transformed into an iProcess procedure. In iProcess, all sub-processes are defined as sub-procedures. Therefore, you have to use sub-procedures instead of activity sets for each process that you wanted to define as a sub-procedure in iProcess.

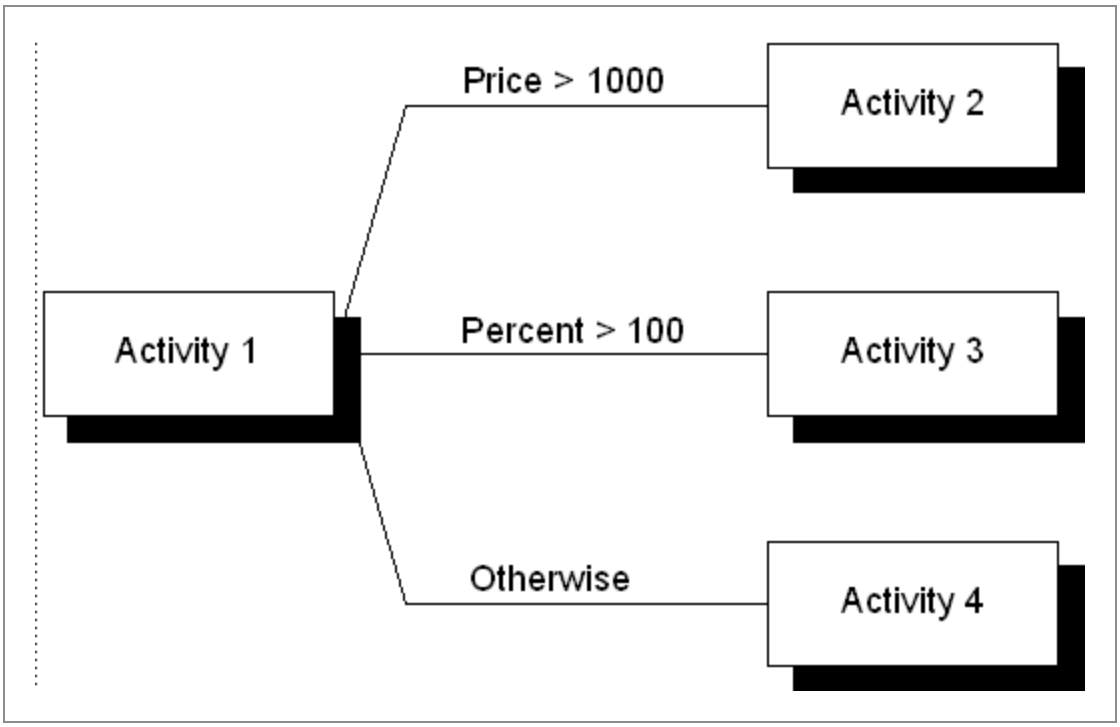
Package Level Applications, Participants and Data Fields

If your XPDL source contains package level applications, participants and data fields, they are ignored when the XPDL source is transformed into an iProcess procedure. This is because iProcess applications, participants and data are defined for each procedure. iProcess has no concept of global data. Therefore, you need to duplicate package level applications, participants and data for each iProcess procedure in the TIBCO iProcess Modeler once you have loaded the XPDL source.

Conditions

Both XPDL and iProcess procedures enable you to process your data from one activity to another depending on a condition. However, XPDL and iProcess handle conditions differently from each other.

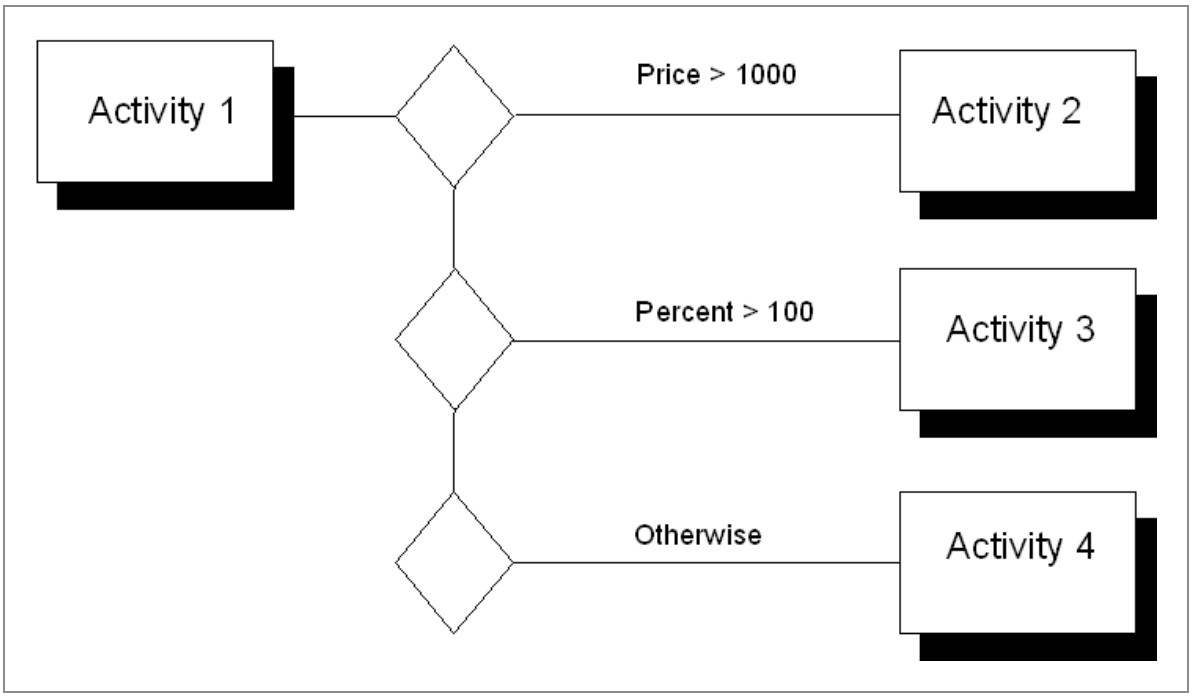
In XPDL, conditions are defined as part of a transition. Each transition can have its own condition and an activity can have multiple branches. The diagram below demonstrates an example of a condition in XPDL:



In this example, when Activity 1 completes then

- Activity 2 is only processed if Price > 1000.
- Activity 3 is only processed if Percent > 100.
- Activity 4 is processed if Activity 2 and Activity 3 are not processed.

iProcess uses a condition object to define conditions. Conditions cannot be defined as part of a link. This means that in iProcess, each step that requires a condition must have its own condition object. The same example above would be represented in an iProcess procedure as follows:



If you are transforming XPDL into an iProcess procedure, you can either:

- in the XPDL source, create a route activity for each condition. iProcess only transforms conditional transitions that come from route activities into an iProcess procedure condition object.

  or

- import the XPDL source into iProcess and amend the procedure so that each condition has a condition object.

If you do not amend the XPDL source to take into account how iProcess handles conditions, then iProcess transforms the XPDL source as follows:

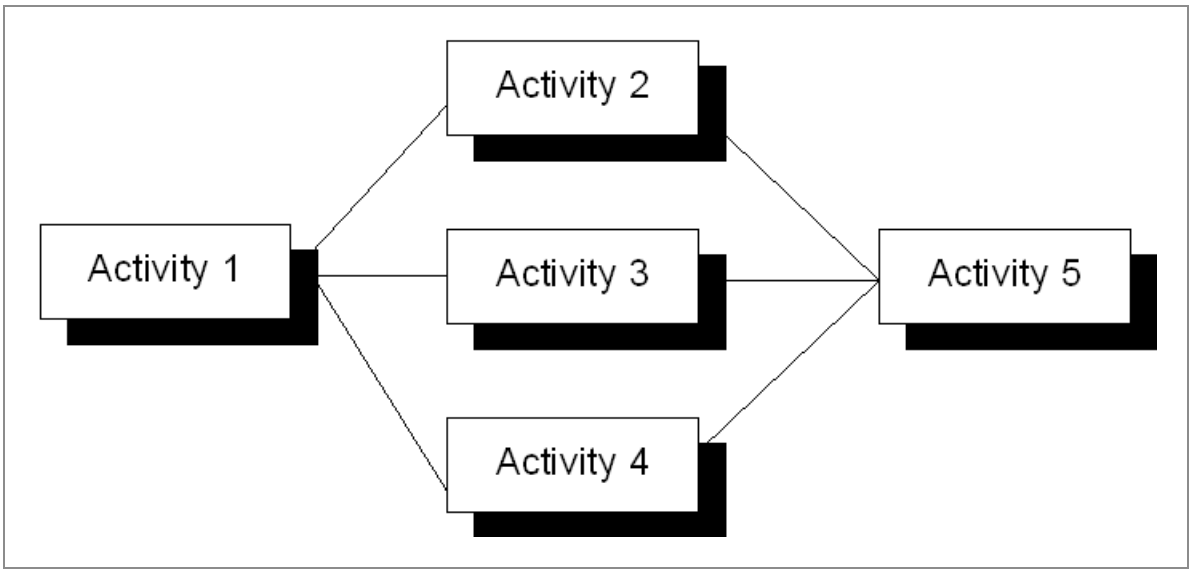
- All conditions from activities other than route activities are ignored. iProcess only transforms conditional transitions that come from route activities into an iProcess procedure condition object in the iProcess procedure.
- If multiple conditional transitions come from a route activity then iProcess transforms the condition from the first transition into an iProcess procedure condition object. All subsequent conditions are created as link labels. This enables you to see the conditions that have not had a condition object created for them so that you can create them later if you wish.
- Conditional transitions into a route object with a join are ignored.

For more information on how iProcess conditions are transformed into XPDL entities, see Understanding XPDL Produced From an iProcess Procedure.

Joining Branches

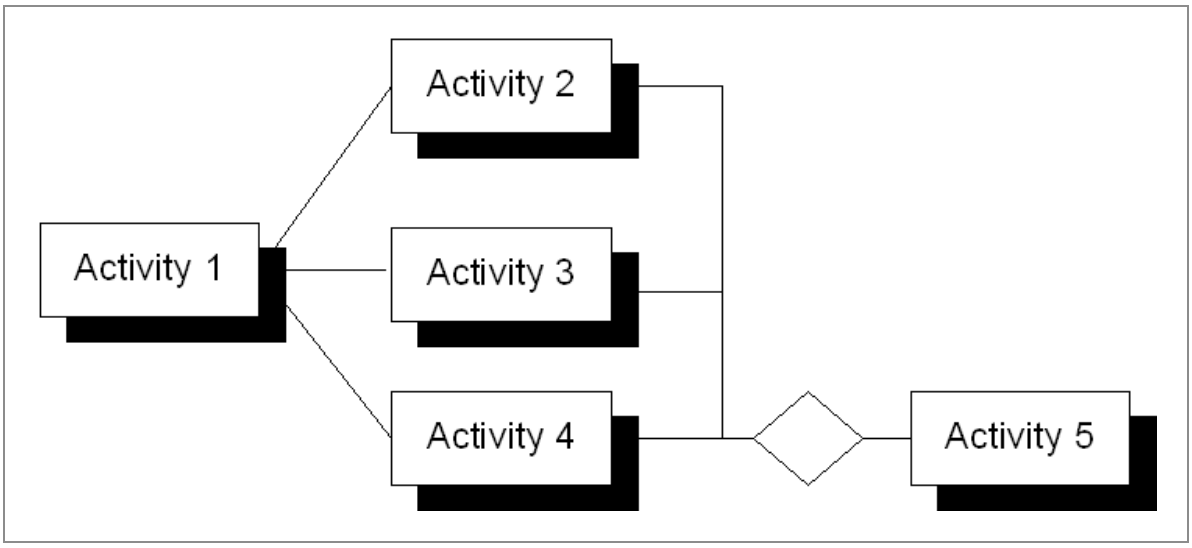
In XPDL and iProcess, you can define a synchronization point in the process where parallel paths join together again. However, XPDL and iProcess handle this differently from each other.

In XPDL, you need to define an AND join as part of a transition between two activities. Each activity can have a join and an activity can have multiple branches. The diagram below demonstrates an example of a join in XPDL:



In this example, when Activity 1 completes then Activity 2, 3 and 4 are processed. Activity 5 does not start processing until Activities 2, 3 and 4 have completed. It is activity 5 that specifies the XPDL and Join.

iProcess uses a wait object to define joins. A join cannot be defined as part of a link. This means that in iProcess, each step that requires a join must have its own wait object. The same example above would be represented in an iProcess procedure as follows:



If you are transforming XPDL into iProcess, you can either,

- in the XPDL source, create a route activity for each And Join. iProcess only transforms route activities with And Joins into an iProcess procedure wait object. In the above XPDL, you would insert a route activity before Activity 5 and specify the And Join on that.

  or

- import the XPDL source into iProcess and amend the procedure accordingly.

If you do not amend the XPDL source to take into account how iProcess handles joins, then iProcess transforms the XPDL source as follows:

- All joins from activities other than route activities are converted to XOR joins. iProcess only transforms join transitions that come from route activities into an iProcess procedure wait object.
- Conditions to and from an AND join are ignored.

For more information on how iProcess Wait objects are transformed into XPDL entities, see Understanding XPDL Produced From an iProcess Procedure.

Naming Conventions

The naming of some iProcess entities is more restrictive than their equivalent entities in XPDL. This means that when transforming XPDL source into an iProcess procedure, if the names of the XPDL entities do not match the naming conventions of the iProcess entities, they are transformed in iProcess as follows:

- any invalid characters are stripped out

- a leading z is placed in front of the name if it starts with a numeric and if that is not allowed in iProcess
- the name is truncated to the length that is required by iProcess for that entity.

For more information about the naming conventions of iProcess entities, see *TIBCO iProcess Modeler Getting Started* and *TIBCO iProcess Modeler Basic Design* guides.

To work around this, you can perform one of the two steps:

- obey the iProcess naming convention when creating your XPDL source
- import the XPDL source into iProcess. Once you have imported the XPDL source into iProcess, you can choose to amend the names or leave them as they are, depending on your requirements.

Note: If you import the XPDL source into iProcess and allow iProcess to truncate the names, you could end up with duplicate names. This could be a problem, for example, if you are referencing a particular step name at run-time and there is more than one step with the same name.

For more information on how iProcess names are transformed into XPDL entities, see Understanding XPDL Produced From an iProcess Procedure.

Formal and Actual Parameters

iProcess has no equivalent concept to FormalParameters and ActualParameters that can be defined in XPDL. In XPDL, FormalParameters and ActualParameters define the input and output parameters to sub-process calls (both the sub-process that is calling and the sub-process that is being called). If you transform an XPDL source that has FormalParameters and ActualParameters defined, iProcess transforms these as follows:

- it checks the FormalParameters and if the name of the FormalParameter matches a data field, it creates the field markings on the iProcess form definition with that name. Otherwise, the FormalParameters are ignored.
- ActualParameters are ignored.

For more information on how iProcess fields are transformed into XPDL entities, see Understanding XPDL Produced From an iProcess Procedure.

Other Vendor Extended Attributes

If the XPDL source you are using has come from another vendor that has defined some extended attributes, these are ignored when you transform the XPDL into an iProcess procedure. To work around this, you can amend the XPDL source so that it transforms the extended attributes into valid iProcess extended attributes. You can do this by performing one of the following steps:

- amending the third party vendor's XPDL using the custom file format facility, see Amending Third-Party Vendor's XPDL.
- manually amend the third party vendor's XPDL so it conforms to the **XPDLExtrAttr.xsd**. For more information, see Using Extended Attributes.

iProcess Procedure Entities That do Not Map to XPDL Entities

The parts of the iProcess procedure that do not map are stored as extended attributes. See Understanding XPDL Produced From an iProcess Procedure.

Transforming Between Different XML Schema Versions

Both XPDL and iProcess procedures have XML schema formats. Each XML schema format has a version number. Inevitably, these schemas will change with later versions as new functionality is added. This means that you must understand how iProcess procedures and XPDL source is transformed when using later versions of the schemas or when using a mix of schema versions.

See:

- About the XPDL XML Schema Format
- About the SchemaFormat Extended Attribute
- Loading XPDL That Uses Earlier Versions of the XML Schema Formats
- Loading XPDL That Uses Later Versions of the XML Schema Versions

About the XPDL XML Schema Format

The version of the XPDL XML schema format that iProcess Workspace (Windows) implements is version 1.0. XPDL source to be loaded into iProcess Workspace (Windows) should conform to this XPDL XML schema format version. For more information about the XPDL and iProcess procedure XML schema format versions, see Transforming Between Different XML Schema Versions.

About the SchemaFormat Extended Attribute

When you save an iProcess procedure as XPDL, the version number of the iProcess procedure XML schema format is defined in the **SchemaFormat** extended attribute. This extended attribute specifies the elements that make up the iProcess procedure XPDL schema format:

- The XPDL XML schema version.
- How generic XPDL elements are mapped to and from procedure elements.
- The iProcess procedure extended attributes schema version.

If the **SchemaFormat** extended attribute is not present when the XPDL document is loaded then it is assumed to be compatible with the current schema format.

Loading XPDL That Uses Earlier Versions of the XML Schema Formats

If you have some XPDL source that has been produced using an earlier XPDL or iProcess XPDL XML schema format, and you want to transform it into an iProcess Workspace (Windows) that uses a later version of the XPDL or iProcess XPDL XML schema format, iProcess Workspace (Windows) automatically upgrades the XPDL source to match the later schema versions.

If the **SchemaFormat** extended attribute is not present when the XPDL document is loaded then it is assumed to be compatible with the current schema format.

Loading XPDL That Uses Later Versions of the XML Schema Versions

If you have some XPDL source that has been produced using a later XPDL or iProcess XPDL XML schema format, and you want to transform it into an iProcess Workspace (Windows) that uses an earlier version of the XPDL or iProcess XPDL XML schema format, then the XPDL will not load. You must re-define your XPDL so that it validates against the correct XML schema format version.

If the **SchemaFormat** extended attribute is not present when the XPDL document is loaded then it is assumed to be compatible with the current schema format.

Amending Third-Party Vendor's XPDL

You can use XSLT to transform your XPDL source to handle any specific functionality that may be unique to a particular third party vendor. To do this, iProcess Workspace (Windows) provides you with a customization facility. It enables you to use a custom file format to transform your XPDL source. You can create some XSLT and specify this as a custom file format. Then, when saving or loading, you can specify that this XSLT file be used to transform the XPDL source.

The source does not necessarily have to be XPDL. The customization facility enables any XML input document to be transformed into a valid iProcess procedure.

See:

- Overview of the Steps Required to Create a Custom File Format
- Creating XSLT to Transform To or From iProcess XPDL
- An Example of an XSLT File Created to Transform XPDL Generated by iProcess for Enhydra JaWE (Java Workflow Editor)
- Creating a Custom File Format

Overview of the Steps Required to Create a Custom File Format

An overview of the steps required to create a custom file format are described below:

1. Create the XSLT file that performs the transformation to or from iProcess XPDL, see Creating XSLT to Transform To or From iProcess XPDL.
2. Create the custom file format configuration file for the **Save As** and/or **Load From** dialog boxes, see Creating a Custom File Format . Custom file formats need to be defined separately for the **Save As** and **Load From** dialog boxes, as different configuration may be required.
3. Select the custom file format from the **Files of Type:** box when loading or saving. See Saving and Loading iProcess Procedures as XPDL.

Creating XSLT to Transform To or From iProcess XPDL

To use the custom file format, you need to create some XSLT to perform the transformation on the XPDL when loading to or saving from iProcess Workspace (Windows). How you create your XSLT depends on the XPDL source you want to transform.

About Creating an XSLT File

If the XSLT file creates an invalid iProcess XPDL document, this will either cause the transformation from XPDL into an iProcess procedure to fail or the loading of the internal procedure XML file will fail. This is because the schema would not validate which would result in a schema validation error.

The most likely cause of this is that the XSLT transformation has placed iProcess XPDL extended attributes in the XPDL that are not valid against the **XPDLExtAtt.xsd** schema. For more information about the extended attributes that can be defined in the XPDL source, see Using Extended Attributes.

An Example of an XSLT File Created to Transform XPDL Generated by iProcess for Enhydra JaWE (Java Workflow Editor)

This section describes an example of custom file format which is an XSLT file that has been created to transform XPDL generated by iProcess for third party vendor, Enhydra JaWE (Java Workflow Editor).

This particular XSLT copies the XPDL produced by the iProcess Workspace (Windows) and adds a **MadeBy** extended attribute to it. Enhydra JaWE uses the same extended attributes to define X and Y co-ordinates of activities on screen. However, JaWE ignores these attributes unless the **Made By** extended attribute is present in the source XPDL. Therefore this transformation adds a **MadeBy = JaWE** attribute to the XPDL, as shown below:

```
<xsl:template match="xpdl:Package/xpdl:ExtendedAttributes">
  <!-- Add the 'made by JAWE extended attributes so that it doesn't ignore the XOffset and
YOffset ext attrs -->
  <xsl:copy>
    <ExtendedAttribute Name="MadeBy" Value="JaWE"/>
    <ExtendedAttribute Name="Version" Value="1.4"/>


    <!-- Then copy everything else underneath -->
    <xsl:apply-templates select="@* | node()" />
  </xsl:copy>
</xsl:template>

</xsl:stylesheet>
```

Displaying Text in the Progress Meter Boxes

When you use the **Load From** and **Save As** dialog boxes, progress meter boxes display the progress of the transformation to XPDL. See Saving and Loading iProcess Procedures as XPDL. Your XSLT can take advantage of these progress meter boxes. You can specify the text that you want to be displayed in one or all of the 3 progress meter boxes. The table below describes the progress meter boxes:

Progress Meter Box	Description
#0	Specifies the text for text box 1

Progress Meter Box	Description
#1	Specifies the text for text box 2
#2	Specifies the text for text box 3

You can specify the text for one, some or all of the progress meter boxes, depending on your requirements.

To specify text for a progress meter box, you need to create an XSL Message element in your XSLT. An example of an **xsl:message** element is shown below:

```
<xsl:message>#0Converting to JaWE XPDL...</xsl:message>
```

Creating a Custom File Format

Separate custom file formats must be defined for the **Save As** and **Load From** dialog boxes, as different configuration may be required depending on whether you are saving or loading. You can create custom file formats for the **Save As** or the **Load From** dialog boxes or for both dialog boxes, depending on your requirements. To create custom file formats for the **Save As** and **Load From** dialog boxes, perform the following steps:

1. Depending on your requirements, create new directories in the following locations:
   - *swclient***PMSaveAs**. For example, if you wanted to transform some XPDL that has been produced from the 3rd party vendor called Enhydra JaWE, you could create a directory called *swclient***PMSaveAs****JaWE**.
   - *swclient***PMLoadFrom**. For example, if you wanted to transform some XPDL that has been created from a 3rd party vendor called Enhydra JaWE, you could create a directory called *swclient***PMLoadFrom****JaWE**.
2. Create your XSLT file and copy it to the directory/directories you created in step 1, depending on your requirements.
3. Navigate to the directory or directories you created in step 1. Using a text editor like **Notepad**, create a new text file in each directory called **transformtype.txt**. The format of the text files should be as follows:

```
<parameter>=<value>
<parameter>=<value>
<parameter>=<value>
```

The valid parameters are:

Parameter	Value	Description
copyfiles	comma separated list of files. For example: **copyfiles=back.gif;forward.gif**	Defines the extra support files that the output file needs at run-time. For example, you may have a list of .**gif** files to support the HTML document. You can specify multiple files in the **copyfiles** parameter. The value of this parameter is a semi-colon separated of files that are to be copied from the transform type's directory into the location for the output file. This is performed before the specified XSLT transformation. The specified copy files can be in sub-directories relative to the transform type directory. In this case, the sub-directory path should be specified. When a sub-directory path is specified, the same directory structure is created in the output location.
description	description of the custom file format. For example: **description=JaWE XPDL**	Defines the text that is: • the description of the custom file format in the **Save As** and **Load From** dialogs. • the description of the files of type in the **Files of Type** drop-down list in the file selection dialogs.

Parameter	Value	Description
exts	defines the file extension. For example: `exts=xpdl;xml`	Defines the valid extensions for this file type. You can define multiple extensions. Each extension must be separated by a semi-colon.
finalfilesource	pathname to final file. For example: **`finalfilesource=frame_data.html`**	This is used with the xsltoutput parameter. It defines the file that is referencing the XSLT output file defined in the xsltoutput parameter.
fullfieldrefs	**yes** or **no**. For example: **`fullfieldrefs=yes`**	By default, when the source XML is created data items that contain references to fields, for example, expressions, scripts and EAI Step Type data, are not transformed into pieces of text and field reference elements. If you want to force full field references in the source XML, enter the following parameter in the **transformtype.txt** file: **fullfieldrefs=yes**
imagedirectory	pathname of image directory. For example: **ImageDirectory=Procedure_Images**	Defines the directory into which the procedure image files are saved. The pathname of the file is relative to the location of the image file. This means that **ImageDirectory=Procedure_Images** is interpreted as **c:\PROCDOC\Procedure_Images**. If no image directory is specified then the image files are created in the same location as the output file.
imagezoom	scale of output images. For example:	Defines the scale of the output procedure images to be set. For

Parameter	Value	Description
	imagezoom=50	example, • **ImageZoom=50**. This means that procedure images will be reduced by 50%. • **ImageZoom=200**. This means that procedure images are enlarged by 200%.
replacefileinfinal	**yes** or **no**. For example: **replacefileinfinal=yes**	Enables the **%FILE%** tag to be used in the finalfilesource parameter.
wantimages	**yes** or **no**. For example: **wantimages=yes**	By default, procedure images are not produced. To produce and output graphical images for each of the procedures, define the **wantimages** parameter, as follows: **wantimages=yes** The image files are created in the same location as the output file. To change this, use the ImageDirectory parameter. The image files are in .png (Portable Network Graphics) format and the format of the name is ***procedurename_Image*.png**. The image files are produced before the XSLT transformation is performed. This means they are available for you to use in the source XML before the XSLT transformation is performed.
wantimagemaps	**yes** or **no**. For example: **wantimagemaps=yes**	By default, image maps are not produced. To produce image map files for each of the procedure images, define the **wantimagemaps** parameter, as

Parameter	Value	Description
		follows: **wantimagemaps=yes** This is used with the wantimages parameter. The wantimages parameter must be set to **yes** to produce the image maps. The image files are produced before the XSLT transformation is performed. This means they are available for you to use in the source XML before the XSLT transformation is performed. The specified XSLT can be coded to make use of image maps to locate and reference individual objects within the procedure image (for example, creating hyperlinks from objects in the procedure image to textual data regarding an object elsewhere in the output file). The image maps are created in the transform type's directory and are deleted on completion of the **Save As** operation. An image map file is output for each procedure and is named according to the procedure name. ***PROCNAME*_ImageMap.xml** The image map files formatted as XML, each object has an **obj** node that contains the following attributes: • id - unique numeric identifier for the object

Parameter	Value	Description
		• name - name (if the object type has a name). • left, right, top, bottom - pixel position of object in image.
xslt	pathname of XSLT output file. For example: `xslt=swxpdljawe.xslt`	Defines the name of the XSLT file that performs the transformation to or from iProcess. The pathname of the file is relative to the location of the **transformtype.txt** file. This means that, as shown in the example below, **swxpdljawe.xslt** is interpreted as **c:\swclient\PMSaveAs\JaWE\swxpdljawe.xslt**.
xsltoutput	pathname to alternative XSLT output file. For example: `xsltoutput=frame_data.html`	Defines an alternative file for the XSLT transformation output. This is parameter can be used if you want your XSLT transformation output to be output to a different file that you don't want you users to see. For example, you may want them to view the XSLT transformation from a file that references the output rather than letting them see the output itself. This parameter is used with the finalfilesource parameter. You can specify a sub-directory relative to the transform type directory. In this case, the sub-directory path should be specified. When a sub-directory path is specified, the same directory structure is created in the output location. You can replace parts of this path using a **%FILE%** tag. For example:

Parameter	Value	Description
		xsltoutput=data\%FILE%_data.html To enable the %FILE% tag, you need to use the replacefileinfinal parameter.

An example of a **transformtype.txt** configuration file for transforming some XPDL from a 3rd party vendor called Enhydra JAWE is shown below:

```
description=JaWE XPDL
exts=xpdl;xml
xslt=swxpdljawe.xslt
```

Saving and Loading iProcess Procedures as XPDL

This section describes how to transform XPDL into iProcess procedures and vice versa. This section describes:

- Loading XPDL into iProcess Workspace (Windows) to create iProcess procedures. You can take any XPDL source and load it into iProcess Workspace (Windows). Once the XPDL has been loaded into iProcess Workspace (Windows), you can amend the procedure as you require - see Loading XPDL into iProcess Workspace (Windows).
- Saving iProcess procedures as XPDL from iProcess Workspace (Windows). Once an iProcess procedure has been saved as XPDL, it can be used with any vendor application that supports XPDL - see Saving a Procedure as XPDL.

Loading XPDL into the iProcess Workspace (Windows)

This section describes how to load XPDL into iProcess Workspace (Windows).

See:

- Before Loading XPDL into iProcess Workspace (Windows)

Before Loading XPDL into iProcess Workspace (Windows)

This section describes some points you need to consider before loading XPDL source into iProcess Workspace (Windows).

- Before loading XPDL source into iProcess Workspace (Windows) you need to understand how procedure version control works.
  - If you import some XPDL for a procedure that does not already exist in iProcess Workspace (Windows), then the procedure is created as version 0.0. This is true, even if the XPDL specifies another version number.
  - If you load some XPDL source for a procedure that already exists in iProcess Workspace (Windows), you must create a new version. You can either create a new minor version or a new major version.

If you create a new minor version number, the procedure is created with the next minor version of the procedure that already exists in iProcess. For example, if the existing procedure has a version of 2.1, iProcess creates the new loaded XPDL as a procedure with a version of 2.2. To do this, from the **Procedure Exists** dialog box, click **OK**.

If you create a new major version number, the major version is taken, either:

from the XPDL if the necessary XPDL extended attribute exists, or

the latest major version on the procedure that already exists in iProcess is used. For example, if the existing procedure has a version of 2.1, iProcess creates the new loaded XPDL as a procedure with a version of 3.0.

To do this, from the **Procedure Exists** dialog box, select **Create new major version**.

- TIBCO does not recommend that you load XPDL for a procedure library into iProcess Workspace (Windows), if the procedure library is saved as XPDL from a later version of TIBCO iProcess Engine and loaded into an earlier version of TIBCO iProcess Engine.

For information about version control, see *TIBCO iProcess Modeler Procedure Management*.

- If the procedure already exists in another library but you would like to import it into a different library, you are prompted to create a shortcut to the procedure in the original library from the new library. To do this, from the **Procedure Exists** dialog box, select **Create shortcut here if procedure is located elsewhere**.

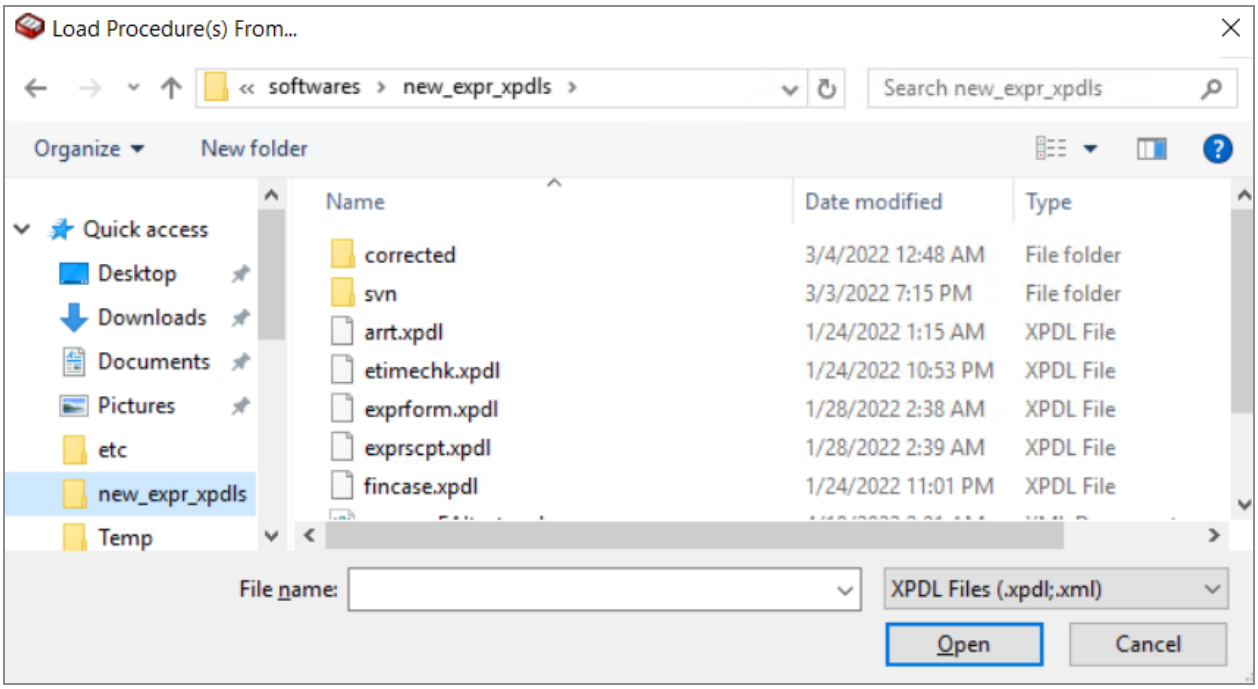
- Before loading XPDL source into iProcess Workspace (Windows) you need to understand the status of the procedures that are created when the XPDL source is loaded. The available options depend on whether or not the procedure is complete:
  - If the procedure is complete, it is loaded with a status of **Unreleased**.
  - If the procedure is incomplete, it is loaded with a status of **Incomplete**.
  - If the procedure already exists and has a status of **Incomplete** or **Unreleased**, then the existing procedure's status is changed to **Withdrawn-Incomplete** or **Withdrawn**, respectively.
- Before loading XPDL into iProcess Workspace (Windows), TIBCO recommend that you set the following extended attributes in the XPDL source:
  - **made by**. This is used to improve load performance in certain areas (for example, activity id's in iProcess procedure XPDL are always output as numeric (and must be numeric in the equivalent procedure object id). If the **MadeBy** attribute is included, the load assumes that activity id's are numeric and therefore no special processing is required in order to ensure that they are converted to numeric).
  - **XOffset** and **YOffset**. If you do not specify the XOffset and YOffset extended attributes the objects are automatically arranged in rows and columns in the order that they appear in the XPDL document rather than in the order that they should be processed.

Loading XPDL into iProcess Workspace (Windows)

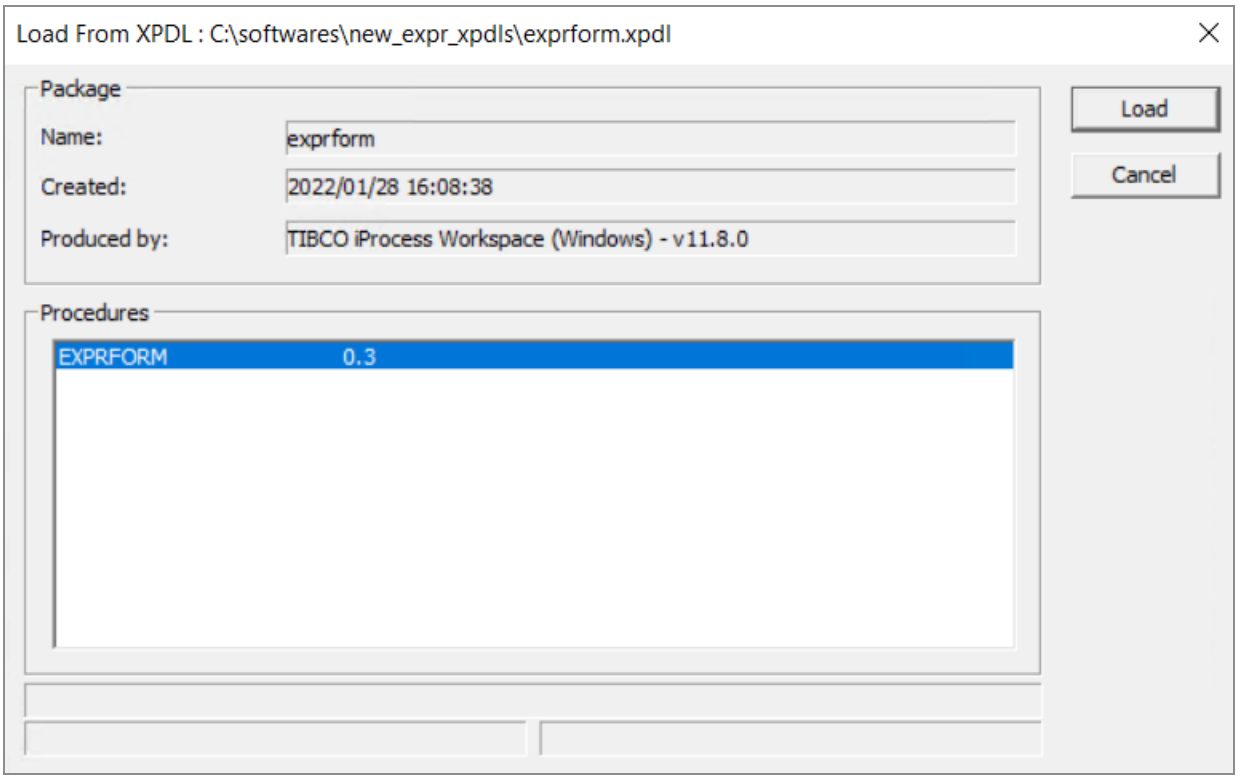
Note: Ensure that the XPDL source to be loaded is in the directory where iProcess Workspace (Windows) is installed.

To load XPDL into iProcess Workspace (Windows), perform the following steps:

1. From the **Procedure Manager**, browse to the procedure library where you want to save the procedure that is to be created from the XPDL.
2. Click **Procedure Management** > **Load From**. The **Load Procedure(s) from...** dialog box is displayed.



3. In the Files of type box, select the file format that you want to use to perform the translation. If you have defined any custom file formats, select the custom file format you want to use from this box. For more information about custom file formats, see Amending Third-Party Vendor's XPDL.
4. Browse to the XPDL file that you want to import.
5. Click **Open**. The **Load From XPDL: *pathname*** dialog box is displayed where *pathname* is the path to the XPDL file you have selected. The **Load From XPDL:** *pathname* dialog box displays information about the **Package** you have selected.



Note: XPDL has the concept of a package. A package acts as a container for a number of different procedures. Therefore, when you are loading some XPDL source, you are loading a package of XPDL source.

6. The **Procedures** box lists the procedures available to be loaded in the **Package** you have selected. Select one, some or all of the procedures, depending on your requirements, and click **Load**.

   A progress meter displays the progress of the transformation to iProcess.

Saving a Procedure as XPDL

This section describes how to save an iProcess procedure as XPDL.

Before Saving a Procedure as XPDL

Before saving a procedure as XPDL, you need to understand which procedure objects can be saved as XPDL:

- You can save the following procedure objects as XPDL:

  - procedure
  - sub-procedure
  - sub-procedure parameter templates.
  - shortcut.

> **Note:** When you save a sub-procedure as XPDL, the referenced I/O template is automatically exported along with the sub-procedure.

- You cannot save a procedure library as XPDL.

See:

- Saving Referenced Sub-Procedures of a Procedure
- Saving a Procedure as XPDL

Saving Referenced Sub-Procedures of a Procedure

When saving a procedure as XPDL, you can decide whether or not to save the referenced sub-procedures of the procedure as XPDL. If you choose to save the referenced sub-procedures of a procedure as XPDL, the referenced I/O templates of the sub-procedures, dynamic sub-procedures, and graft steps are automatically exported together.

To configure this option for all saving operations, follow these steps:

1. Highlight the **Procedure Manager** panel from iProcess Workspace (Windows). Then select **Options** > **Windows Options...**. The Procedure Management Configuration dialog box is displayed.
2. To save the referenced sub-procedures, check the **Save Referenced Subprocedures** check box from the **Load From / Save As** section of the dialog box. Click **OK**. Otherwise, uncheck the check box and click **OK**.

After you make this configuration, you can still choose to save or not to save the sub-procedures in each saving operation. See the following section for details.

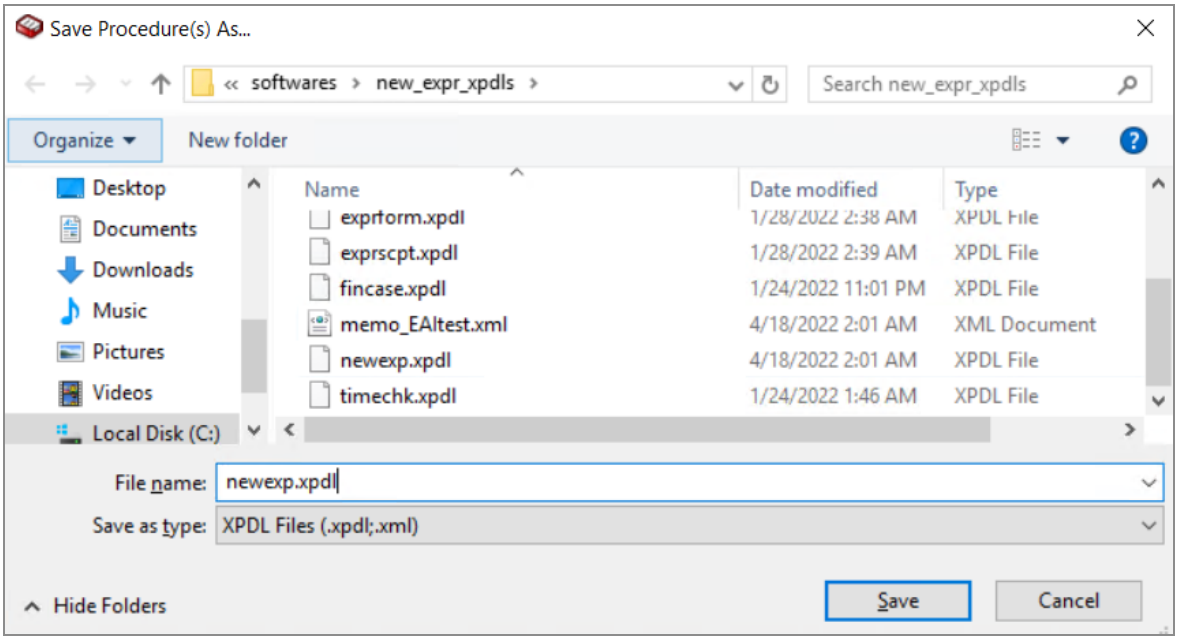
Saving a Procedure as XPDL

To save a procedure as XPDL, complete the following steps:

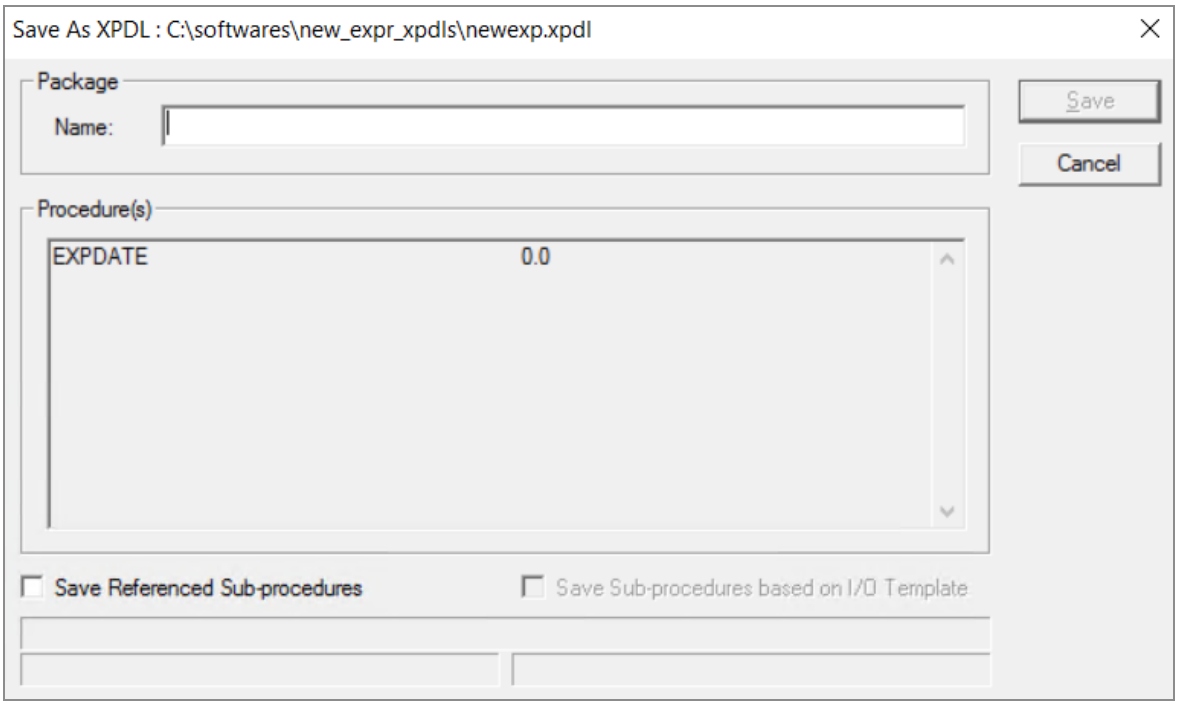
1. From the **Procedure Manager**, browse to the location of the iProcess procedure(s)

that you want to save as XPDL.

2. Select the procedure(s) that you want to save as XPDL.
3. Click **Procedure Management** > **Save As XPDL...**. The **Save Procedure(s) As...** dialog box is displayed.



4. Navigate to the directory where you want to save the XPDL file.
5. In the Save as type: box, select the file format that you want to use to perform the translation. If you have defined any custom file formats, select the custom file format you want to use from this box. For more information about custom file formats, see Amending Third-Party Vendor's XPDL.
6. In the File name: box, enter the name you want to call the procedure to be saved as XPDL.
7. Click **Save**. The Save As XPDL: *pathname* dialog box is displayed where *pathname* is the path to the XPDL file you have selected.



8. In the **Name** field, enter the name for the XPDL package.
9. Check the **Save Referenced Sub-procedures** check box if you want to save the sub-procedures.

   Note: If you check this check box to save sub-procedures, the referenced I/O templates of the sub-procedures, dynamic sub-procedures and graft step are automatically exported together.

   The configuration made through this check box takes precedence over the **Save Referenced Sub-procedures** check box in the Procedure Management Configuration dialog box. In the Procedure Management Configuration dialog box, you can choose to save or not to save the sub-procedures for all procedures. You can customize each saving operation with this check box in the Save As XPDL dialog box.
10. Check the **Save Sub-procedures based on I/O Template** check box if you want to save all the sub-procedures based on the I/O templates, which are referenced by the sub-procedures.

    This check box is enabled when the **Save Referenced Sub-Procedures** check box is checked.
11. Click **Save**.

> **Note:** XPDL has the concept of a package. A package acts as a container for a number of different procedures. Therefore, when you are saving an iProcess procedure as XPDL, you are saving it as a package of XPDL.

A progress meter displays the progress of the transformation to XPDL.

Interpreting the Logs Produced From the Transformation Process

This section describes the log files that are produced from the transformation process.

See:

- Logging the Transformation Process
- Understanding Progress Logs
- Understanding Errors Loading XPDL into iProcess Workspace (Windows)

Logging the Transformation Process

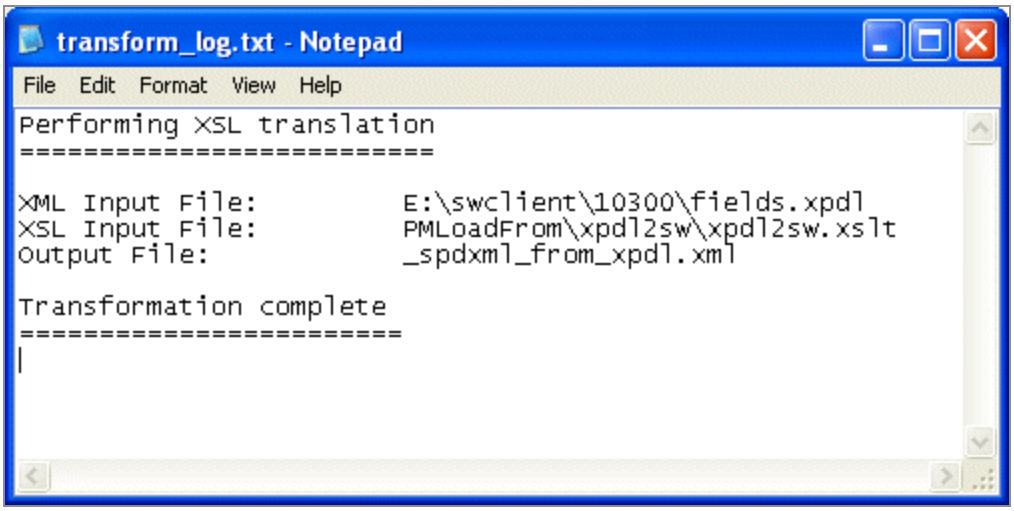
To log the transformation process, do the following:

1. Highlight the **Procedure Manager** panel from iProcess Workspace (Windows). Then select **Options** > **Windows Options...**. The **Procedure Management Configuration** dialog box is displayed.
2. Check the **Full transformation logging** check box from the **Load From / Save As** section of the dialog box. Click **OK**.

Understanding Progress Logs

The *swclient***xslTransform_log.txt** logs the progress of each transformation. The file is overwritten each time a **Load From** or **Save As** operation is performed. The file logs all messages, for example, if the transformation was successful or if errors were reported during the process.

An example of a *swclient***xslTransform_log.txt** file is shown below:



Understanding Errors Loading XPDL into iProcess Workspace (Windows)

If there are any errors or warnings generated in the translation process, these are processed as follows:

- The errors and/or warnings are reported to the *swclient***xslTransform_log.txt**. You can choose to view the log file at this point, or open it in a text editor later.
- the XML input or output source stream is output to the following files, depending on whether you are loading from or saving to iProcess Workspace (Windows):
  - *swclient***_loadfrom_spdxml.xml**
  - *swclient***_saveas_spdxml.xml**

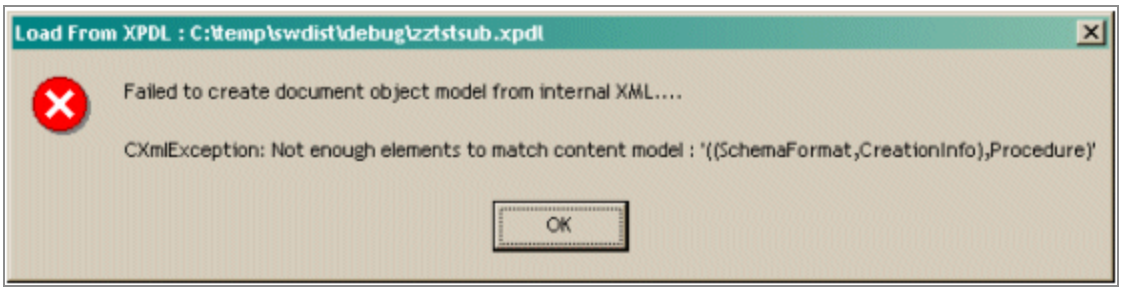
If any errors are generated by the translation process, TIBCO recommend that you take copies of these files and contact TIBCO Support.

Loading Invalid XPDL

If the XPDL you are trying to load is invalid or contains no valid procedure, the loading operation may fail and you will receive an error message similar to the one below:

```
Failed to create document object model from internal XML...
```

```
CXmlException: Not enough elements to match content model : ’
((SchemaFormat,CreationInfo),Procedure)’
```



If you receive an error like this you should check that your XPDL is valid and matches the iProcess XPDL XML schema format. For more information about the iProcess XPDL XML schema format, see Understanding XPDL Produced From an iProcess Procedure.

Using the CUTIL Utility to Export Procedures and Call Hierarchies of Procedures

This section describes how to use the CUTIL utility to export procedures or XML reports about hierarchical call tree of procedures.

Overview of the CUTIL Utility

The CUTIL utility, which is located in the *IPWW_HOME* directory, provides the ability to export procedures and hierarchical call tree of procedures. You can perform the following activities:

- Set up the CUTIL utility connection. You need to set up the connection before exporting procedures or call hierarchies of procedures.
- Export procedures and procedure libraries.
- Generate hierarchical reports for procedures, sub-procedures, sub-procedure parameter templates, and procedure libraries.

Command-Line Parameters and Options

The following table lists all the options and parameters for the CUTIL utility and short descriptions of them.

Command-Line Parameters and Options	Description
The following commands are used to set up the CUTIL utility connection.	
cutil.exe <hostname> <rpcnumber> <username> [password]	Connects to an iProcess Engine server by using the command-line options.

Command-Line Parameters and Options	Description
cutil.exe -ws <servername> <hostname> <rpcnumber>	Adds or updates configurations of an iProcess Engine server in the `servers.cfg` file.
cutil.exe -wu <servername> <username>	Adds or updates configurations of users for an existing iProcess Engine server without a password in the `servers.cfg` file.
cutil.exe -wU <servername> <username> [password]	Adds or updates configurations of users for an existing iProcess Engine server with a password in the `servers.cfg` file.
cutil.exe -o <ALL|servername>	View all user configurations for one or all servers.
cutil.exe -r <servername> <username> [password]	Connects to an iProcess Engine server by using the connection details from the `servers.cfg` file.
cutil.exe ?	Displays the `CUTIL` command-line parameters and options for setting up a utility connection.
After successfully setting up the CUTIL utility connection, a prompt is displayed. You can run the following commands to export procedures or output call hierarchies for procedures at the command prompt.	
?	Displays all the `CUTIL` utility based commands.
EXPORT <procedure,procedure,...> [export path] <options>	Exports procedures to XPDL files.
EXPORTLIB <library path, library path,...|/> [export path] <options>	Exports procedure libraries to XPDL files.
CALLTREE <procedure|template|library> [version|-sN] <-c|-p|-a> [-o path] [-f file name]	Generates a hierarchical XML report based on a procedure, sub-procedure, sub-procedure parameter template, or procedure library.

Command-Line Parameters and Options	Description
`quit`	Terminates the connection to an iProcess Engine server.

Setting Up the CUTIL Connection

To set up the `CUTIL` connection, you need to use the `CUTIL` utility to connect to an iProcess Engine server from the client side. You must set up the connection before exporting procedures or exporting hierarchical call tree of procedures.

This section describes the following topics:

- General Usage of the CUTIL Utility
- Connecting to iProcess Engine Servers
- Return Values in Scripts

General Usage of the CUTIL Utility

Type the following command to view the `CUTIL` command-line parameters and options for setting up a utility connection:

```
cutil.exe ?
```

Connecting to iProcess Engine Servers

The `CUTIL` utility uses parameters and options to give information about iProcess Engine servers and users that would normally be specified as part of the server connections.

Note: Only the iProcess users with the `PRODEF` or `ADMIN` MENUNAME attribute are allowed to log in iProcess Engine servers by using the `CUTIL` utility. For more information about the MENUNAME attributes, see "Setting Pre-defined Attributes" in *TIBCO Workspace (Windows) Manager's Guide*.

Two ways to connect to iProcess Engine servers:

- Run the `CUTIL` utility with parameters and options, see Connecting to iProcess Engine Servers by Using the Command-Line Options.
- Run the `CUTIL` utility with the configured connection details from the `servers.cfg` file, see Connecting to an iProcess Engine Server by Using the Configuration File.

  To use the `servers.cfg` file for connecting to an iProcess Engine server, you must configure the server and user information in the file:

  — Add or update configurations of iProcess Engine servers in the `servers.cfg` file, see Adding or Updating Server Information to the Configuration File.

  — Add or update configurations of users for iProcess Engine servers in the `servers.cfg` file, see Adding or Updating User Information to the Configuration File.

  — Display all user configurations for one or all servers, see Viewing the Configured Information of Servers and Users.

Connecting to iProcess Engine Servers by Using the Command-Line Options

To connect to an iProcess Engine server by using the command-line options, type the following command:

`cutil.exe` *<hostname> <rpcnumber> <username> [password]*

where:

- *hostname* is the host name or IP address of an iProcess Engine server that you want to connect to.
- *rpcnumber* is the RPC number of an iProcess Engine server. You can find it in the *SWDIR*`\swdefs` file.
- *username* is the name of the user that you want to connect to an iProcess Engine server.
- *password* is the user's password that you use to connect to an iProcess Engine server. Additionally, if you omit the password when typing the command, the `CUTIL` utility prompts you for the password and obscures the text you typed with asterisks (*).

Note: If you type valid values for the options, the specified user will be connected to the iProcess Engine server, but the server and user information are not recorded in the `servers.cfg` configuration file.

If you connect to the iProcess Engine server successfully, the following prompt will be displayed:

```
enter '?' for help
>
```

Example

To connect to a server with the host name, `ipe1141`, and user name, `user1`, and password, `123`, enter the following command:

```
C:\TibcoiProcessWorkspace>cutil.exe ipe1141 391875 user1 123
```

If the password is not given on the command prompt, a prompt will be displayed to remind you to input the password:

```
C:\TibcoiProcessWorkspace>cutil.exe ipe1141 391875 user1
Please input the password:
>
```

Then you must input the password.

Adding or Updating Server Information to the Configuration File

To add new or update existing configurations of an iProcess Engine server in the `servers.cfg` file, type the following command:

`cutil.exe -ws` <*servername*> <*hostname*> <*rpcnumber*>

where:

- *servername* is the name of an iProcess Engine server.
- *hostname* is the host name or IP address of the iProcess Engine server that you want to connect to.

- *rpcnumber* is the RPC number of the iProcess Engine server. You can find it in the *SWDIR*\swdefs file.

If you input valid values for the options, the server information will be added in or updated to the servers.cfg file.

Example

The following example writes information about an iProcess Engine server to the servers.cfg file:

```
C:\TibcoiProcessWorkspace>cutil.exe -ws server1 ipe1141 391875
The information of the server 'SERVER1' was added successfully.
Server Name Host Address RPC Number User Name Password Supplied
-----------------------------------------------------------------
SERVER1      ipe1141      397815
```

Adding or Updating User Information to the Configuration File

To add new or update existing user information in the servers.cfg file:

- Enter the following command without a password:

  cutil.exe -wu <*servername*> <*username*>

- Enter the following command with a password:

  cutil.exe -wU <*servername*> <*username*> *[password]*

where:

- *servername* is the name of an iProcess Engine server.
- *username* is the name of the user that you want to connect to the iProcess Engine server.
- *password* is the user's password that you use to connect to the iProcess Engine server. Additionally, if you omit the password when typing the command, the CUTIL utility prompts you for the password and obscures the text you typed with asterisks (*).

Example

- The following example shows how to write a user's information for server1 to the servers.cfg file without a password:

```
C:\TibcoiProcessWorkspace>cutil.exe -wu server1 user1
The user 'user1' for the server 'SERVER1' was added
   successfully.
Server Name Host Address RPC Number User Name Password Supplied
-----------------------------------------------------------------
 SERVER1     ipe1141       397815     user1       No
```

- The following example shows how to write a user's information for `server1` to the `servers.cfg` file with a password:

  ```
  C:\TibcoiProcessWorkspace>cutil.exe -wU server1 user1 123
  The user 'user1' for the server 'SERVER1' was updated successfully.
  Server Name Host Address RPC Number User Name Password Supplied
  -----------------------------------------------------------------
   SERVER1     ipe1141       397815     user1       Yes
  ```

  If the password is not given on the command line, a prompt will be displayed to remind you to enter the password:

  ```
  C:\TibcoiProcessWorkspace>cutil.exe -wU server1 user1
  Please Input the password:
  >***
  The user 'user1' for the server 'SERVER1' was updated successfully.
  Server Name Host Address RPC Number User Name Password Supplied
  -----------------------------------------------------------------
   SERVER1     ipe1141       397815     user1       Yes
  ```

Connecting to an iProcess Engine Server by Using the Configuration File

To connect to an iProcess Engine server by using the connection details from the `servers.cfg` file, type the following command at a command prompt:

`cutil.exe -r` <*servername*> <*username*> *[password]*

where:

- *servername* is the name of an iProcess Engine server you configured in the `servers.cfg` file.
- *username* is the name of the user that you want to connect to the iProcess Engine server. The user name you entered can be either configured or unconfigured in the `servers.cfg` file.

- *password* is the user's password that you use to connect to an iProcess Engine server. The password you entered can be either configured or unconfigured in the `servers.cfg` file.
  - If the password is not configured in the `servers.cfg` file, you must run the `cutil.exe` command with the password. If you omit the password, the `CUTIL` utility prompts you for the password and obscures the text you typed with asterisks (*).
  - If the password is saved in the `servers.cfg` file and you run the `cutil.exe` command with the password, the password you entered on a command prompt will be used first. If the password is not valid, the password in the configuration file will be used instead.

If you connect to the iProcess Engine server successfully, the following prompt will be displayed:

```
enter '?' for help
>
```

Example

To connect to the `server1` using the information from the `servers.cfg` file, enter the following command:

- If the password is configured in the `servers.cfg` file:

  ```
  C:\TibcoiProcessWorkspace>cutil.exe -r server1 user1
  ```

- If the password is not configured in the `servers.cfg` file:

  ```
  C:\TibcoiProcessWorkspace>cutil.exe -r server1 user1 123
  ```

  If the password is not given on the command prompt, a prompt will be displayed to remind you to input the password:

  ```
  C:\TibcoiProcessWorkspace>cutil.exe -r server1 user1
  Please input the password:
  >
  ```

  Then you must input the password.

Viewing the Configured Information of Servers and Users

To view the configured information about servers and users, run the following command:

`cutil.exe -o <ALL|servername>`

where:

- `ALL` displays all the information about servers and users.
- *servername* is the name of an iProcess Engine server you want to view.

Example

To view configuration of `server1`, enter the following command:

```
C:\TibcoiProcessWorkspace>cutil.exe -o server1
Server Name Host Address RPC Number User Name Password Supplied
-----------------------------------------------------------------
 SERVER1     ipe1141      397815     user1         No
 SERVER1     ipe1141      397815    SWADMIN        Yes
 SERVER1     ipe1141      397815     SWPRO         No
 SERVER1     ipe1141      397815     SWPRO1        No
```

Return Values in Scripts

If you run the `CUTIL` utility to set up a connection from a script, the return values from the utility and their descriptions are shown in the following table.

Value	Description
1	Success
-1	Access Denied
-2	Internal error
-4	Invalid parameters
-11	Unable to find the configured server by the server name

Value	Description
-22	Invalid host name or RPC number
-51	Invalid user name or password

Generating Reports for Procedures, Templates, and Libraries

After successfully connecting to an iProcess Engine server and being displayed with a prompt, you can generate an XML report for a procedure, sub-procedure, sub-procedure parameter template, or procedure library by using the `CALLTREE` command within the `CUTIL` utility. This section explains the following topics:

- Introduction to the XML Report
- Using the CALLTREE Command to Generate Reports

Introduction to the XML Report

The generated XML report demonstrates a hierarchical call tree of a procedure, sub-procedure, sub-procedure parameter template, or procedure library. It helps you view the structure and relationships of the procedures, templates, and libraries. The following sections describe:

- Naming Rules for the XML Report
- Setting the Location of the XML Report
- Definition of the XML Report

Naming Rules for the XML Report

This section explains how to name an XML report:

- Rules for Report Names
- Specifying the Report Name

Rules for Report Names

The default name used for the generated XML report is as follows:

objectname`-`*M*`-`*m*`-`*option*`.xml`

where:

- *objectname* is the name of the object, which can be a procedure, a template, or a library.
- *M* is the major version number of the procedure or template.
- *m* is the minor version number of the procedure or template.
- *option* is the content that are extracted, which can be all, children, or parents.

For example:

- The default name of a report for parent objects of the `proc` procedure with version 1.0:

  `proc-1-0-parents.xml`
- The default name of a report for child objects of the `temp` template with version 1.0:

  `temp-1-0-children.xml`
- The default name of a report for all objects of the lib library:

  `lib-all.xml`

Specifying the Report Name

You can specify the generated XML report by running the `CALLTREE` command with the `-f` *file name* option.

For example:

```
CALLTREE tempf -c -f tempf
```

This command uses `-f tempf` to name the report as `tempf.xml`.

For more information, see Syntax of the CALLTREE Command.

Setting the Location of the XML Report

The generated XML report is saved in the *IPWW_HOME* directory by default. You can change the directory by running the following command:

CALLTREE <*procedure*|*template*|*library*> <-c|-p|-a> -o *path*

where -o *path* specifies where you want to save the XML report.

For example:

```
CALLTREE proc -c -o c:\bin
```

This command uses -o c:\bin to specify saving the report in the c:\bin directory.

For more information about CALLTREE command parameters and options, see Syntax of the CALLTREE Command.

Definition of the XML Report

The format of the XML report must conform to the CallTree.xsd schema, which is located in the *IPWW_HOME* directory. Therefore, you can review the CallTree.xsd file to understand the definition of the XML report.

To customize your XML report, run the CALLTREE command with command-line parameters and options. For more information, see Output of the CALLTREE Command.

Using the CALLTREE Command to Generate Reports

Run the CALLTREE command to generate a hierarchical call tree of a procedure, sub-procedure, sub-procedure parameter template, or procedure library in an XML report. You can use the command-line parameters and options to customize the report.

Before generating an XML report, you must connect to an iProcess Engine server. For more information, see Setting Up the CUTIL Connection.

Note:
Only the iProcess users with the PRODEF or ADMIN MENUNAME attribute and the Use, View, or Edit access type are allowed to administer procedures by using the CUTIL utility.

For more information about the MENUNAME attributes, see "Setting Pre-defined Attributes" in *TIBCO Workspace (Windows) Manager's Guide*. For more information about the access type, see "Setting Access Controls" in *TIBCO iProcess Modeler Procedure Management*.

If you connect to the iProcess Engine server successfully, the following prompt will be displayed:

```
enter '?' for help
>
```

You can enter the question mark (?) to view all the CUTIL utility based commands.

Syntax of the CALLTREE Command

The CALLTREE command with command-line parameters and options are as follows:

CALLTREE <*procedure*|*template*|*library*> [*version*|-sN] <-c|-p|-a> [-o *path*] [-f *file name*]

where:

- *procedure* is the name of a procedure or a sub-procedure.
- *template* is the name of a sub-procedure parameter template.
- *library* is the name of a library.
- *version* specifies the version of a procedure or a sub-procedure parameter template that you want to display in the report. Specify one of the following parameters:
  - -r lists the call tree for the released version.
  - -u lists the call tree for the unreleased version.
  - -m lists the call tree for the Model version.
  - -v*X*.*Y* lists the call tree for the version *X*.*Y*, where *X* is the major version number and *Y* is the minor version number.

  If the value of *version* is not specified on a command line, the default precedence of selecting procedure versions is:

  Released version > Unreleased version > Model.
- -sN (optional) specifies the precedence that are used to select the version of sub-procedures called by its caller. Rules are as follows.

Value	Precedence
1	Released version only

Value	Precedence
2	Unreleased version> Released version
3	Model > Released version
4	Unreleased version > Model > Released version
5	Model >Unreleased version > Released version

If the `-sN` flag is not specified, the precedence of procedure versions set in TIBCO iProcess Workspace (Windows) will be applied. For more information about precedence settings in TIBCO iProcess Workspace (Windows), see "Defining Which Version of a Sub-Procedure is Called" in *TIBCO iProcess Modeler Procedure Management*.

Note:

When generating a hierarchical call tree of a library, the following order is used rather than setting the `-sN` option to select the version of procedures or templates called by the library:

`rumiwxn`

where:

- `r` specifies the current released version.
- `u` specifies the current unreleased version.
- `m` specifies the current Model version.
- `i` specifies the current incomplete version.
- `w` specifies the latest withdrawn version.
- `x` specifies the latest incomplete or withdrawn version.
- `n` specifies no preference.

- `-c` specifies the child objects of a procedure, template, or library. For more information, see Output of the CALLTREE Command.
- `-p` specifies the parent objects of a procedure, template, or library. For more information, see Output of the CALLTREE Command.

- `-a` specifies the child and parent objects of a procedure, template, or library. For more information, see Output of the CALLTREE Command.
- `-o` *path* specifies the location where you want to save the XML report. For more information, see Setting the Location of the XML Report.
- `-f` *file name* specifies the name of the XML report. For more information, see Naming Rules for the XML Report.

Output of the CALLTREE Command

An XML report is generated after running the `CALLTREE` command, but the contents of the report vary depending on the command-line parameters and options. You can use the following table as a reference to customize your XML report.

Command	Main Objects in the Output Call Tree
`CALLTREE` *procedure* `[`*version*`|-sN] -c`	The call tree lists: • All the sub-procedures with the specified *version* that are called directly and indirectly from *procedure*. • The templates used by the dynamic sub-procedure steps or graft steps that are in the specified *version* of *procedure*. • Properties of the procedure, sub-procedures, and templates. • Input and output parameters of the sub-procedures and templates.
`CALLTREE` *procedure* `[`*version*`|-sN] -p`	The call tree lists: • All the libraries that contain the specified *version* of *procedure*. • The templates that are used by the sub-procedures. • All the sub-procedures that call the specified *version* of *procedure*. • Properties of the procedures, libraries, and templates. • Input and output parameters of the sub-procedures and templates.
`CALLTREE` *procedure* `[`*version*`|-sN] -a`	The call tree lists all the output of the `CALLTREE` *procedure* `[`*version*`|-sN] -c` and `CALLTREE` *procedure* `[`*version*`|-sN] -p` commands.

Command	Main Objects in the Output Call Tree
CALLTREE *template* [version|-sN] -c	The call tree lists: • All the sub-procedures that are based on the template, and all the child objects of these sub-procedures. • Properties of the template and sub-procedures.
CALLTREE *template* [version|-sN] -p	The call tree lists: • All the libraries that contain the specified *version* of *template*. • All the procedures that contain the dynamic sub-procedure steps and graft steps using the specified *version* of *template*, and all the parent objects of these procedures. • Properties of template, libraries, and procedures. • Input and output parameters of the sub-procedures.
CALLTREE *template* [version|-sN] -a	The call tree lists all the output of the CALLTREE *template* [version|-sN] -c and CALLTREE *template* [version|-sN] -p commands.
CALLTREE *library* -c	The call tree lists: • All the procedures, templates, and libraries that are included in *library*. • Properties of the procedures, templates, and libraries.
CALLTREE *library* -p	The call tree lists: • The ancestor libraries of *library*. • Properties of the libraries.
CALLTREE *library* -a	The call tree lists all the output of the CALLTREE *library* -c and CALLTREE *library* -p commands.

Return Values in Scripts

If you run the CALLTREE command from a script, the return values from the command and their descriptions are shown in the following table.

Value	Description
1	Success
-1	Access denied
-2	Internal error
-4	Invalid parameters
-10	Failed to allocate the memory
-11	Unable to find the entity referenced by the -sN option

Exporting Procedures and Procedure Libraries to XPDL Files

This section describes how to export procedures or procedure libraries to a single XPDL file by using the EXPORT or EXPORTLIB command within the CUTIL utility.

Before exporting procedures or procedure libraries, you must connect to an iProcess server first. For more information, see Setting Up the CUTIL Connection.

Note:
Only the iProcess users with the PRODEF or ADMIN MENUNAME attribute and the Use, View, or Edit access type are allowed to administer procedures by using the CUTIL utility.

For more information about the MENUNAME attributes, see "Setting Pre-defined Attributes" in *TIBCO Workspace (Windows) Manager's Guide*. For more information about the access type, see "Setting Access Controls" in *TIBCO iProcess Modeler Procedure Management*.

If you connect to the iProcess Engine server successfully, the following prompt is displayed:

```
enter '?' for help
>
```

You can enter the question mark (?) to view all the `CUTIL` utility based commands.

Exporting Procedures to XPDL Files

After successfully connecting an iProcess Engine server and being displayed with a prompt, you can export procedures to an XPDL file by running the `EXPORT` command with command-line parameters and options:

`EXPORT` <*procedure*,*procedure*,...> [*export path*] <*options*>

where:

- *procedure*,*procedure*,... specifies the name of the procedures you want to export. You can export one or more procedures to a single XPDL file.

Note: When you save a sub-procedure as XPDL, the referenced I/O template is automatically exported along with the sub-procedure.

- *export path* (optional) specifies where you want to save the exported XPDL file. The default directory is *IPWW_HOME*.
- *options* can be one or more of the following to suppress or change a specific option.

Option	Description
`-e`	Do not prompt for the version of the procedure to export. Use the current default version of the procedure.
`+e` *version* /*precedence*	Do not prompt for the version of the procedure to export. Use the specified version or precedence value, for example, `+e 3.0` or `+e rumiwxn`. For more information about precedence, see *TIBCO iProcess Modeler Procedure Management*.
`-s`	Do not export referenced sub-procedures.
`+s`	Export referenced sub-procedures along with the procedure.

Option	Description
	Note: If you choose to save the referenced sub-procedures of a procedure as XPDL, the referenced I/O templates of the sub-procedures, dynamic sub-procedures, and graft steps are automatically exported together.
`-x`	Do not prompt for the version of the sub-procedures to export.
`+x` *precedence*	Do not prompt for the version of the sub-procedure to export and use the supplied precedence value.
`-f`	Do not prompt for the file name to save the XPDL file. Use the first procedure name in the input command as the file name.
`+f` *file name*	Do not prompt for the file name to save the XPDL file. Use the specified file name. **Note**: File naming limitations: • The name must be 28 characters or less in length. • The name may contain letters, numbers, and underscore characters.
`-p`	Do not prompt for package name to export. Use the default package name, `ProcessPackage`.
`+p` *package name*	Do not prompt for package name to export. Use the specified package name. **Note**: Package naming limitations: • The name must be 64 characters or less in length. • The name may contain letters, numbers, and underscore characters.
-t	Do not export the sub-procedures based on the I/O templates, which are referenced by the sub-procedures of the main procedure.

Option	Description
	Note: This option should be used with the +s option.
+t	Export all the sub-procedures based on the I/O templates, which are referenced by the sub-procedures of the main procedure. **Note:** This option should be used with the +s option.

Examples of Exporting Procedures

The following examples show how to use the EXPORT command with the command-line parameters and options:

- To export the proc1 and proc2 procedures with the ur precedence, enter the following command:

```
>EXPORT proc1,proc2 +e ur -sfp
```

  The proc1.xpdl file is generated in the *IPWW_HOME* directory.

- To export the proc1 procedure with the ur precedence and all the sub-procedures of proc1 with the urm precedence, enter the following command. The package name is specified as pck1, and the name of the exported XPDL file is specified as file1.

```
>EXPORT proc1 +e ur +s +x urm +f file1 +p pck1
```

  The file1.xpdl file is generated in the *IPWW_HOME* directory.

Return Values in Scripts

If you run the EXPORT command from a script, the return values from the command and their descriptions are shown in the following table.

Value	Description
1	Success

Value	Description
0	Export aborted using 'q'/'Q'
-1	Access denied
-2	Internal error
-4	Invalid parameters
-10	Failed to allocate the memory
-11	Unable to find the entity referenced

Exporting Procedure Libraries to XPDL Files

After successfully connecting to an iProcess Engine server and being displayed with a prompt, you can export procedure libraries to a single XPDL file by running the `EXPORTLIB` command with command-line parameters and options:

`EXPORTLIB` <*library path*, *library path*, ... | /> [*export path*] <*options*>

where:

- *library path*, *library path*, ... specifies the full path name of libraries as displayed in the Procedure Manager. You can export one or more libraries to a single XPDL file.

  For example, to export the `/dept/purchasing` library, enter the following command:

  ```
  EXPORTLIB /dept/purchasing
  ```

- / specifies the root library. For example, to export the root library, enter the following command:

  ```
  EXPORTLIB /
  ```

- *export path* (optional) specifies where you want to save the exported XPDL file. The default directory is *IPWW_HOME*.
- *options* can be one or more of the following to suppress or change a specific option.

Option	Description
`-e`	Do not prompt for the version of the procedure to export. Use the current default version of the procedure.
`+e` *version /precedence*	Do not prompt for the version of the procedure to export. Use the specified version or precedence value, for example, `+e 3.0` or `+e rumiwxn`. For more information about precedence, see *TIBCO iProcess Modeler Procedure Management*.
`-f`	Do not prompt for the file name to save the XPDL file. Use the first library name in the input command as the file name.
`+f` *file name*	Do not prompt for the file name to save the XPDL file. Use the specified file name. **Note**: File naming limitations: • The name must be 28 characters or less in length. • The name may contain letters, numbers, and underscore characters.
`-p`	Do not prompt for package name to export. Use the default package name, `ProcessPackage`.
`+p` *package name*	Do not prompt for package name to export. Use the specified package name. **Note**: Package naming limitations: • The name must be 64 characters or less in length. • The name may contain letters, numbers, and underscore characters.
`-l`	Do not export the objects in the sub-libraries.
`+l`	Export the referenced objects in the sub-libraries.

Examples of Exporting Procedure Libraries

The following examples show how to use the EXPORTLIB command with the command-line parameters and options:

- To export the /lib/sublib library and use file1 as the XPDL file name, enter the following command:

  ```
  >EXPORTLIB /lib/sublib +f file1 -epl
  ```

  The file1.xpdl file is generated in the *IPWW_HOME* directory.

- To export the /lib/sublib library with the ur precedence and all the objects in the sub-libraries, enter the following command. The package name is specified as pck1, and the name of the exported XPDL file is specified as file1.

  ```
  >EXPORTLIB /lib/sublib +e ur +f file1 +p pck1 +l
  ```

  The file1.xpdl file is generated in the *IPWW_HOME* directory.

Return Values in Scripts

If you run the EXPORTLIB command from a script, the return values from the command and their descriptions are shown in the following table.

Value	Description
1	Success
0	Export aborted using 'q'/'Q'
-1	Access denied
-2	Internal error
-4	Invalid parameters, for example, invalid export path
-10	Failed to allocate the memory
-11	Unable to find the library

Understanding XPDL Produced From an iProcess Procedure

Once you have transformed your iProcess procedure into XPDL, you need to understand the XPDL that is produced.

How iProcess Transforms to XPDL

The following table describes how the iProcess entities are transformed into XPDL entities and extended attributes.

iProcess	XPDL	Extended Attributes
Procedure package	Package	• MadeBy • SchemaFormat
Procedure	XPDL WorkflowProcess	• ProcProperties • LayoutOptions • NonFlowObject • Link • SwimLanes
Sub-Procedure I/O Parameters	WorkflowProcess/Formal Parameters	• SubProcParams within the ProcProperties extended attribute
Field	Workflow/Process/DataFields/DataField	• FieldDetails
Work Item Step Form	WorkflowProcess/ Applications/Application	• FormDef

iProcess	XPDL	Extended Attributes
Definitions		
EAI Step Type Specific Definitions	WorkflowProcess/ Applications	• EAIRunType • EAIData
Complex Router	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = ComplexRouterObject • StepNum • Annotation • TextPosition
Condition Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = ConditionObject • CondPredict • Annotation • TextPosition
Dynamic Sub-Procedure Call	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = SubprocObject • StepNum • Annotation • TextPosition • Deadline

iProcess	XPDL	Extended Attributes
		• PredictDuration • DynStepFlags • SubProcNameArray • StartStepArray • Template • InputMappings • OutputMappings • DynGraftError
EAI Step	Activity with Type Implementation/Tool/ Application	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = EAIObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • EAIStepFlags • AuditInitiated • AuditComplete • DelayedRelease
Event Step	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= EventObject • StepNum

iProcess	XPDL	Extended Attributes
		• Annotation • TextPosition • Deadline • PredictDuration
Graft Step	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = GraftStepObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • GraftStepFlags • ReturnSubProcName • Template • OutputMappings • DynGraftError
Start Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = StartObject
Normal Step	Activity with Type Implementation/Tool/ Application	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = StepObject

iProcess	XPDL	Extended Attributes
		• StepNum • Annotation • TextPosition • Deadline • PredictDuration • StepFlags • ExtendedDescription • StepCommands • StepPriority • Addressees
Stop Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = StopObject
Sub-Procedure Call Step	Activity with Type Implementation/SubFlow	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= DynamicSubprocObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • SubPStepFlags • SubProcedureDetails

iProcess	XPDL	Extended Attributes
		• SubProcCallParams
Transaction Control Step	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= TCStepObject • StepNum • Annotation • TextPosition • Deadline • ControlType
Wait Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= WaitObject
Link	Transition	• LinkFlags • ColorRef • LinkStyleFlags • Label • Elbows
Withdraw Step Link	WorkflowProcess/ ExtendedAttributes	• Link
Annotation	Extended Attribute/ExtAttr/ NonFlowObject with element = Annotation Object	• NonFlowObject • Annotation
EIS Report	Extended Attribute/ExtAttr/ NonFlowObject with element = EISObject	• NonFlowObject • EISObject

iProcess	XPDL	Extended Attributes
Link Router (Elbow)	ExtendedAttribute/ExtAttr/ NonFlowObject with element - ElbowObject	• NonFlowObject • Elbows
Script	ExtendedAttribute/ExtAttr/ NonFlowObject with element = ScriptObject	• NonFlowObject

The following table describes how the iProcess entities are transformed into XPDL entities and extended attributes.

iProcess	XPDL	Extended Attributes
Procedure package	Package	• MadeBy • SchemaFormat
Procedure	XPDL WorkflowProcess	• ProcProperties • LayoutOptions • NonFlowObject • Link • SwimLanes
Sub-Procedure I/O Parameters	WorkflowProcess/Formal Parameters	• SubProcParams within the ProcProperties extended attribute
Field	Workflow/Process/DataFields/DataField	• FieldDetails
Work Item Step Form Definitions	WorkflowProcess/ Applications/Application	• FormDef
EAI Step Type Specific	WorkflowProcess/ Applications	• EAIRunType • EAIData

iProcess	XPDL	Extended Attributes
Definitions		
Complex Router	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = ComplexRouterObject • StepNum • Annotation • TextPosition
Condition Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = ConditionObject • CondPredict • Annotation • TextPosition
Dynamic Sub-Procedure Call	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = SubprocObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • DynStepFlags

iProcess	XPDL	Extended Attributes
		• SubProcNameArray • StartStepArray • Template • InputMappings • OutputMappings • DynGraftError
EAI Step	Activity with Type Implementation/Tool/ Application	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = EAIObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • EAIStepFlags • AuditInitiated • AuditComplete • DelayedRelease
Event Step	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= EventObject • StepNum • Annotation • TextPosition

iProcess	XPDL	Extended Attributes
		• Deadline • PredictDuration
Graft Step	Activity with Type Implementation/No	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = GraftStepObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • GraftStepFlags • ReturnSubProcName • Template • OutputMappings • DynGraftError
Start Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = StartObject
Normal Step	Activity with Type Implementation/Tool/ Application	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = StepObject • StepNum • Annotation

iProcess	XPDL	Extended Attributes
		• TextPosition • Deadline • PredictDuration • StepFlags • ExtendedDescription • StepCommands • StepPriority • Addressees
Stop Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType = StopObject
Sub-Procedure Call Step	Activity with Type Implementation/SubFlow	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= DynamicSubprocObject • StepNum • Annotation • TextPosition • Deadline • PredictDuration • SubPStepFlags • SubProcedureDetails • SubProcCallParams
Transaction	Activity with Type Implementation/No	• XOffset = X co-ordinate

iProcess	XPDL	Extended Attributes
Control Step		• YOffset = Y co-ordinate • ObjType= TCStepObject • StepNum • Annotation • TextPosition • Deadline • ControlType
Wait Object	Activity with Type Route	• XOffset = X co-ordinate • YOffset = Y co-ordinate • ObjType= WaitObject
Link	Transition	• LinkFlags • ColorRef • LinkStyleFlags • Label • Elbows
Withdraw Step Link	WorkflowProcess/ ExtendedAttriubtes	• Link
Annotation	Extended Attribute/ExtAttr/ NonFlowObject with element = Annotation Object	• NonFlowObject • Annotation
EIS Report	Extended Attribute/ExtAttr/ NonFlowObject with element = EISObject	• NonFlowObject • EISObject
Link Router (Elbow)	ExtendedAttribute/ExtAttr/ NonFlowObject with element -	• NonFlowObject • Elbows

iProcess	XPDL	Extended Attributes
	ElbowObject	
Script	ExtendedAttribute/ExtAttr/ NonFlowObject with element = ScriptObject	• NonFlowObject

About XPDL Entities Transformed From iProcess

This section describes how the elements that make up each iProcess entity are transformed into the elements that make up the XPDL entity. For example, the ID and name for an XPDL WorkflowProcess entity is derived from the procedure name of the iProcess entity.

See:

- Procedure Package
- Procedure
- Sub-Procedure Input/Output Parameters
- Field
- Normal Step Form Definition
- EAI Step Type Definition
- Complex Router
- Condition
- Dynamic Sub-Procedure Call
- EAI Step
- Event Step
- Graft Step
- Start Step

- Normal Step
- Stop Object
- Sub-Procedure Call Step
- Transaction Control Step
- Wait Step
- Link
- Annotation
- EIS Report
- Link Router
- Script

Procedure Package

XPDL has the concept of a procedure package. There is no equivalent concept in iProcess. When an iProcess procedure is transformed into XPDL, the following iProcess extended attributes become part of an XPDL procedure package entity:

- MadeBy – Tibco-Process-Suite.
- SchemaFormaSchemaForma – Schema Format Version information.

Procedure

An iProcess procedure maps to an XPDL WorkflowProcess. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Description
Procedure Name	WorkflowProcess ID and Name	The ID and Name of the XPDL WorkflowProcess
	WorkflowProcess	Can be either PUBLIC or PRIVATE

iProcess Element	XPDL Element	XPDL Element Description
	Access Level	• PUBLIC = Main procedure • PRIVATE = Sub-procedure or Sub-procedure I/O Parameter Template

Note: There is no equivalent of an iProcess I/O Parameter Template in XPDL. These are stored as an XPDL WorkflowProcess with no activities or transitions. The ProcedureProcedure extended attribute can be used to distinguish I/O parameter templates from other procedures. For example, ProcType element = IOTEMPLATE rather than MAIN or SUB.

Sub-Procedure Input/Output Parameters

iProcess Sub-procedure input/output parameters are mapped to XPDL WorkflowProcess Formal Parameters. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Descriptions
Sub-Procedure Parameter ID	FormalParameter ID	FormalParameter ID is the sub-procedure parameter ID appended by either **ip** or **op** • **ip** for input • **op** for output
	FormalParameter Mode	FormalParameter Mode is either IN or OUT • IN is for input parameters • OUT is for output parameters

Note:

- iProcess sub-procedure I/O Parameters are separated into input and output parameter sections (in XPDL all parameters are defined in one section and referenced according to their sequence).
- iProcess sub-procedure I/O Parameters have unique IDs within their section. This means you may have a parameter in the input section with the same ID as a parameter in the output section.
- When transferred to XPDL, parameter IDs are appended with **ip** (input parameter) or **op** (output parameter) to ensure that they are unique within the XPDL parameter definitions.
- The complete I/O parameter definitions are also stored within the Sub-Procedure Input/Output ParametersSub-Procedure Input/Output Parameters extended attribute as only a representation can be stored within generic XPDL.

Field

An iProcess field maps to an XPDL WorkflowProcess DataFields or DataField. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Descriptions
Field Name	ID and Name	The ID and name of the XPDL field
	IsArray	IsArray is either YES or NO depending on whether or not the field is an array
Field Length	Length	The length of the field
Field Data Type	Data Type	Data Type is one of the following: • STRING - text/memo fields • FLOAT - numeric fields

iProcess Element	XPDL Element	XPDL Element Descriptions
		• DATETIME - date fields • DATETIME - time fields • REFERENCE - All other fields

Normal Step Form Definition

An iProcess normal step field definition maps to an XPDL WorkflowProcess/Applications/Application. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Descriptions
Work Item Step Name	ID	
	Name	is **TIBCO-Process-Suite-WorkItem** which indicates the application to process the form
Field Markings	FormalParameters	For each field marking on the form definition a FormalParameter is output. The FormalParameters are made up as follows: • ID = Fieldname and Unique ID. This is to ensure that there are not duplicate IDs where two or more instances of the same field appear on the form • Mode = Field Marking Type. It can be one of the following: • IN = DISPLAY/EMBEDDED • OUT = CALCULATED/HIDDEN • INOUT = REQUIRED /OPTIONAL

iProcess Element	XPDL Element	XPDL Element Descriptions
		• DataType = is one of the following: • STRING - text/memo fields • FLOAT - numeric fields • DATETIME - date fields • DATETIME - time fields • REFERENCE - All other fields

EAI Step Type Definition

An iProcess EAI Step Type Definition maps to an XPDL WorkflowProcess/Applications/Application. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element
EAI Step Name	ID
EAI Type Name	Name

Note: No FormalParameters are output.

Complex Router

An iProcess complex router maps to an XPDL activity with Type Route. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element
Numeric Object ID	ID
Complex Router Name	Name

Condition

An iProcess condition maps to an XPDL activity with Type Route. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element
ID	Numeric Object ID	
	Name	is **Condition_Router**

Note:

- The Condition expression is placed on all Transition/Condition's from the XPDL activity that are created from outgoing 'true' links.
- All false links from the condition object are defined as 'otherwise' transitions from the XPDL activity.

Dynamic Sub-Procedure Call

An iProcess dynamic sub-procedure call maps to an XPDL Activity with Type Implementation/No. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Description
Numeric Object ID	ID	

iProcess Element	XPDL Element	XPDL Element Description
Dynamic sub-procedure call name	Name	
	StartMode	is Automatic
	FinishMode	is Automatic
Deadline	Deadline	The deadline is made up as follows: • Execution is either SYNCHR or ASYNCHR: — SYNCHR = Withdraw on expire — ASYNCHR = Do not withdraw • Condition is the textual representation of the deadline expressions/period • Exception name is the unique reference to the deadline transition ID in the following format: **DeadlineExpired_trans**_*uniqueid* **Note:** The complete deadline is also stored as an extended attribute.

EAI Step

An iProcess EAI Step maps to an XPDL activity with Type Implementation/Tool/Application. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Description
Numeric Object ID	ID	

iProcess Element	XPDL Element	XPDL Element Description
EAI Step Name	Name	
	StartMode	
	FinishMode	
	Deadline	The deadline is made up as follows: • Execution is either SYNCHR or ASYNCHR: — SYNCHR = Withdraw on expire — ASYNCHR = Do not withdraw • Condition is the textual representation of the deadline expressions/period • Exception name is the unique reference to the deadline transition ID in the following format : **DeadlineExpired_trans_***uniqueid* **Note:** The complete deadline is also stored as an extended attribute.
	Implementation type.	The implementation type is made up as follows: • Tool[Type = APPLICATION]. The EAI Type specific definition data is stored as an XPDL WorkflowProcess/Applications/Application • Tool Id = EAI step name • ActualParameters = unused (all fields used in application are references to procedure field definitions). For more information on EAI type specific definitions in XPDL Applications see EAI Step Type Specific DefinitionsEAI Step Type Specific Definitions

iProcess Element	XPDL Element	XPDL Element Description
		• Performer = <EAIStepName>_Performer. This references an XPDL WorkflowProcess/Participants/Participant that specifies the first Addressee of the step. (XPDL only allows a single participant definition)

Event Step

An iProcess event step maps to an XPDL activity with type implementation/no. The following table shows the transformation details for this entity

iProcess Element	XPDL Element	XPDL Element Description
Numeric Object ID	ID	
Event step name	Name	
	StartMode.	The StartMode is configured as follows: • Automatic if the event step is part of the workflow. For example, if the event is processed as an action of another step and halts the branch until the event is triggered • Manual if the event is not part of the workflow. For example, it can be triggered asynchronously
	FinishMode.	The FinishMode is Manual. For example, a user or external application triggers it
	Implementation type	The implementation type is made up as follows: • Tool[Type = APPLICATION]. The EAI Type specific

iProcess Element	XPDL Element	XPDL Element Description
		definition data is stored as an XPDL WorkflowProcess/Applications/Application • Tool Id = EAI step name • ActualParameters = unused (all fields used in application are references to procedure field definitions). For more information on EAI type specific definitions in XPDL Applications see EAI Step Type Specific DefinitionsEAI Step Type Specific Definitions • Performer = <EAIStepName>_Performer. This references an XPDL WorkflowProcess/Participants/Participant that specifies the first Addressee of the step (XPDL only allows a single participant definition)

Graft Step

An iProcess graft step maps to an activity with type implementation/no. The following table shows the transformation details for this entity

iProcess Element	XPDL Element	XPDL Element Description
Numeric Object ID	ID	
Dynamic Sub-procedure Call name	Name	
	StartMode	is Automatic

iProcess Element	XPDL Element	XPDL Element Description
	FinishMode	is Automatic
	Deadline.	The deadline is made up as follows: • Execution is either SYNCHR or ASYNCHR: — SYNCHR = Withdraw on expire — ASYNCHR = Do not withdraw • Condition is the textual representation of the deadline expressions/period • Exception name is the unique reference to the deadline transition ID in the following format : **DeadlineExpired_trans**_*uniqueid* **Note:** The complete deadline is also stored as an extended attribute.

Start Step

An iProcess step maps to an XPDL activity with type route. A route activity with a single transition to default first activity in process. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element
Numeric Object ID	ID
	Name is **Start**

Normal Step

An iProcess normal step maps to an activity with type implementation/tool/application. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Description
Numeric Object ID	ID	
Normal step name	Name	
	StartMode	The StartMode is Automatic. The work item is automatically delivered to a work queue by the iProcess Engine.
	FinishMode	The FinishMode is Manual. The work item is manually released by a user.
	Deadline	The deadline is made up as follows: • Execution is either SYNCHR or ASYNCHR: — SYNCHR = Withdraw on expire — ASYNCHR = Do not withdraw • Condition is the textual representation of the deadline expressions/period • Exception name is the unique reference to the deadline transition ID in the following format : **DeadlineExpired_trans_***uniqueid* **Note:** The complete deadline is also stored as an extended attribute.
	Implementation type	The implementation type is made up as follows:

iProcess Element	XPDL Element	XPDL Element Description
		• Tool[Type = APPLICATION]. The EAI Type specific definition data is stored as an XPDL WorkflowProcess/Applications/Application • Tool Id = Normal Step Name • ActualParameters = Fields referenced by form definition mapped onto the Application defined for the form associated with this step. For more information, see Work Item Step Form DefinitionsWork Item Step Form Definitions. • Performer = *StepName_Performer* where: — *StepName* is the name you defined for your Step *Performer* references an XPDL WorkflowProcess/Participants/Participant that specifies the first Addressee of the step (XPDL only allows a single participant definition)
	WorkflowProcess/Participants/Participant	The WorkflowProcess/Participants/Participant is configured as follows: • Id is the ***step name_Performer*** (links to Activity/Performer) • Name is the iProcess user/role/field name • ParticipantType is configured as

iProcess Element	XPDL Element	XPDL Element Description
		follows: — HUMAN = iProcess User/Group addressee (or “sw_starter” for user who initiated case (procedure instance)) — ROLE = iProcess Role addressee — RESOURCE = Data field addressee **Note:** As only the first addressee is transferable into generic XPDL, the complete Addressee definitions are also stored as an extended attribute.

Stop Object

An iProcess stop object maps to an XPDL activity with type route. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element
Numeric Object ID	ID
	Name is **Stop**

Note:

- This is a route activity with a single transition from the last step in the process branch.
- This object is optional and is inferred by an object with no output links.

Sub-Procedure Call Step

An iProcess Sub-Procedure Call Step maps to an XPDL Activity with Type Implementation/Subflow. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element
Numeric Object ID	ID	
Sub-Procedure Call step name	Name	
	StartMode	is Automatic. The sub-procedure is automatically called by the iProcess Engine.
	FinishMode	is Automatic. The sub-procedure is automatically called by the iProcess Engine.
	Deadline	The deadline is made up as follows: • Execution is either SYNCHR or ASYNCHR: — SYNCHR = Withdraw on expire — ASYNCHR = Do not withdraw • Condition is the textual representation of the deadline expressions/period • Exception name is the unique reference to the deadline transition ID in the following format : **DeadlineExpired_ trans**_*uniqueid* **Note:** The complete deadline is also stored as an extended attribute.
Sub-Procedure Name	SubFlow ID	

iProcess Element	XPDL Element	XPDL Element
	Subflow Execution	Subflow Execution is always SYNCHR. iProcess procedures do support calling a sub-process asynchronously. To configure this, you need to define a branch in the procedure and call out the sub-process on the branch.
The expressions (where a valid expression can be simply a reference to a field) mapped onto the sub-process I/O parameters	SubFlow Actual Parameters	**Note:** the complete parameter mappings are also stored as extended attributes. This is because in an iProcess procedure these mappings are made via unique Id's for each parameter whereas XPDL relies on the correct sequencing of parameters.

Transaction Control Step

An iProcess transaction control step maps to an XPDL Activity with Type Implementation/No. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element Description
Numeric Object ID	ID	
Transaction Control Step Name	Name	
	StartMode	is Automatic
	FinishMode	is Automatic

Wait Step

A wait step is the iProcess procedure equivalent of an XPDL 'AND Join' (i.e. waits for all steps on incoming procedure branches to complete before continuing along a single branch). All other joins (multiple transitions into a single activity) are treated as 'XOR

Joins'. A wait step maps to an XPDL Activity with Type Implementation/No. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element
Numeric Object ID	ID
	Name is **Wait_Router**

Link

An iProcess link maps to an XPDL Transition. The following table shows the transformation details for this entity.

iProcess Element	XPDL Element	XPDL Element
	ID .	is **trans**_*uniqueid* where *uniqueid* is an automatically generated unique ID
	From/To	From/To are the object IDs that the link connects from and to
Link label text	Description	
	Condition type	The condition type is configured as follows: • CONDITION. This contains the contents of the condition expression if the link is a link from a condition step's true action. • OTHERWISE. If the link is

iProcess Element	XPDL Element	XPDL Element
		from a condition step's false action, an OTHERWISE transition type is created. Multiple false links create multiple OTHERWISE transitions. • EXCEPTION. If the Deadline link contents of the Condition reference to the 'from' activity's Deadline/ExceptionName value (i.e. "DeadlineExpired_Trans_ **uniqueid**").
	Activity/TransitionRestrictions/Transition Restriction	These are defined only when there are multiple links to and from a single object. They are configured as follows: • Multiple links from an object are created as AND types. This is because iProcess uses nested routing object constructs to define complex conditional paths. • Multiple links to an object are created as XOR types. If the object that is being linked to is a Wait step then it is created as an AND type.

Note: This does not include Withdraw Step Links. These are stored as WorkflowProcess level Extended Attributes.

Annotation

An iProcess annotation maps to an XPDL ExtendedAttribute/ExtAttr/NonFlow Object with sub-element AnnotationObject.

EIS Report

An iProcess EIS report maps to an XPDL ExtendedAttribute/ExtAttr/NonFlow Object with sub-element EISObject.

Link Router

An iProcess link router maps to an XPDL ExtendedAttribute/ExtAttr/NonFlow Object with sub-element ElbowObject.

Script

An iProcess script maps to an XPDL ExtendedAttribute/ExtAttr/NonFlow Object with sub-element ScriptObject.

Using Extended Attributes

XPDL has the concept of extended attributes. Extended attributes can be defined by the user or vendor to capture any additional entity characteristics that need to be exchanged between systems. Entities in iProcess that do not have an equivalent in XPDL are saved as extended attributes during translation. This section describes how to use extended attributes between iProcess and XPDL.

See:

- About iProcess Extended Attributes
- Types of iProcess Extended Attributes
- About the XPDLExtrAttr.XSD Schema Format
- Understanding the iProcess Extended Attributes

About iProcess Extended Attributes

Entities in iProcess that do not have an equivalent in XPDL are saved as extended attributes. The benefit of this is that you do not lose parts of the procedure. This is especially true if you are transforming an iProcess procedure more than once. For example, if you transform an iProcess procedure into XPDL and then transform it back into iProcess, you could lose parts of the iProcess procedure if they were not defined as extended attributes.

Extended attributes are intended to be used by any user or vendor who wants to capture additional entity characteristics. This means that other XPDL vendors could name their extended attributes with the same name as an iProcess extended attribute. To overcome this, iProcess extended attributes are defined within an **ExtAttr** element from the iProcess namespace. For example:

```
<ExtendedAttribute Name="ExtAttr">
  <ExtAttr xmlns="http://bpm.tibco.com/2004/SWSPDXML2.0" Name="FieldDetails">
     <FieldDetails fsize="10">
          <FieldType>DATE</FieldType>
     </FieldDetails>
  </ExtAttr>
</ExtendedAttribute>
```

This means that the TIBCO iProcess Workspace (Windows) can distinguish between iProcess procedure extended attributes and other vendor extended attributes even if they have the same names.

Types of iProcess Extended Attributes

There are two types of extended attribute definitions:

- an extended attribute that contains an XML element definition:

```
<ExtendedAttribute Name="ExtAttr">
  <ExtAttr xmlns="http://bpm.tibco.com/2004/SWSPDXML2.0" Name="FieldDetails">
     <FieldDetails fsize="10">
          <FieldType>DATE</FieldType>
     </FieldDetails>
  </ExtAttr>
</ExtendedAttribute>
```

- an extended attribute that is defined as a simple name value pair:

```
<ExtendedAttribute Name="ExtAttr">
  <ExtAttr xmlns="http://bpm.tibco.com/2004/SWSPDXML2.0" Name="ObjType" Value="StepObject"/>
</ExtendedAttribute>
```

About the XPDLExtrAttr.XSD Schema Format

When you install the TIBCO iProcess Workspace (Windows), the **XPDLExtrAttr.xsd** schema is copied to the directory where you installed iProcess Workspace (Windows).

If your XPDL source contains some iProcess Extended Attributes, then when you load the XPDL source into the TIBCO iProcess Workspace (Windows), it is validated against the **XPDLExtrAttr.xsd** schema. If the extended attributes are not valid, the XPDL does not load.

Understanding the iProcess Extended Attributes

The following table describes the extended attributes that are used by iProcess:

Extended Attribute	Description
Addressees	Defines the addressee or addressees to whom the step is sent.
AlternateIcon	Defines the alternative icon if an alternative icon has been specified.
Annotation	Defines the text that documents the iProcess entities.
AuditComplete	Defines the expression for the value of an **EAI Step Call-Out Completed** message in a case audit trail.
AuditInitiated	Defines the expression for the value of an **EAI Step Call-Out Initiated** message in a case audit trail.
ColorRef	Defines the link color.
CondPredict	Defines how to handle conditions during case prediction.

Extended Attribute	Description
ControlType	Defines the transaction control type.
Deadline	Defines the deadline.
DelayedRelease	Defines delayed release settings.
Description	Defines the object description.
Duration	Defines the duration of the step between the step being active and released.
DynGraftError	Defines error handling properties.
DynStepFlags	Defines step property flags.
EAIData	Defines EAI type specific step definition data.
EAIRunType	Defines EAI type for run-time processing because this may be different from the EAI type at design-time.
EAIStepFlags	Defines EAI step property flags.
EAITypeData	Defines the EAI type data because this may be different from the EAIData.
Elbows	Defines elbow-joint objects on the link.
Expression	Defines the expression.
ExtAttr	The element that defines iProcess extended attributes.
ExtendedDescription	Defines the second step description.
FieldDetails	Defines the original field type/length /format specification.
FieldMark	Defines the field marking.

Extended Attribute	Description
FieldRef	Defines the unique ID of a field reference.
FieldType	Defines the field type.
FormDef	Defines complete form layout definition.
GraftStepFlags	Defines graft step property flags.
InputMappings	Defines input parameter mappings.
Label	Defines any labels.
LayoutOptions	Defines the default layout/link style options.
Link	Defines the workflow process links that have no XPDL equivalent (see WithdrawStepLink for more information).
LinkStyleFlags	Defines extra link style flags.
LinkFlags	Defines extra link connection properties.
NonFlowObject	Defines procedure objects not involved in the process (see non-workflow objects for more information).
OldSubProcParam	Defines the sub-procedure field mappings.
OutputMappings	Defines output parameter mappings.
PredictDuration	Defines predicted duration.
ProcProperties	Defines procedure properties.
PublicStep	Defines any public steps.
ReturnSubProcName	Defines the array data field for return of grafted sub-procedure names.

Extended Attribute	Description
RoleName	Defines the roles.
SchemaFormat	Defines the version of the iProcess Extended Attributes schema you are using. If it is not present when the XPDL document is loaded then it is assumed to be compatible with the schema format currently employed by the loading software.
Script	Defines any scripts.
ScriptRef	Defines the unique ID of a script.
StartStepArray	Defines date field for sub-procedure start steps.
StepCommands	Defines the initial/keep/release command expressions.
StepFlags	Defines the step property flags.
StepPriority	Defines the work item priority settings.
SubProcCallParams	Defines the details of the sub-procedure I/O parameter mappings.
SubProcedureDetails	Defines details of the sub-procedure to call.
SubProcNameArray	Defines data fields for sub-procedure names.
SubPStepFlags	Defines step property flags.
SwimLanes	Defines the details of the swim lanes.
Template	Defines reference to the sub-procedure I/O parameter template procedure used by this object.
TextPosition	Defines the annotation label position.
UserName	Defines the name of the user.

Extended Attribute	Description
XPDLExtAttr_XSD_Version	Defines the version of the iProcess Extended Attributes XSD.

The following table describes the extended attributes that are used by iProcess:

Extended Attribute	Description
Addressees	Defines the addressee or addressees to whom the step is sent.
AlternateIcon	Defines the alternative icon if an alternative icon has been specified.
Annotation	Defines the text that documents the iProcess entities.
AuditComplete	Defines the expression for the value of an **EAI Step Call-Out Completed** message in a case audit trail.
AuditInitiated	Defines the expression for the value of an **EAI Step Call-Out Initiated** message in a case audit trail.
ColorRef	Defines the link color.
CondPredict	Defines how to handle conditions during case prediction.
ControlType	Defines the transaction control type.
Deadline	Defines the deadline.
DelayedRelease	Defines delayed release settings.
Description	Defines the object description.
Duration	Defines the duration of the step between the step being active and released.
DynGraftError	Defines error handling properties.

Extended Attribute	Description
DynStepFlags	Defines step property flags.
EAIData	Defines EAI type specific step definition data.
EAIRunType	Defines EAI type for run-time processing because this may be different from the EAI type at design-time.
EAIStepFlags	Defines EAI step property flags.
EAITypeData	Defines the EAI type data because this may be different from the EAIData.
Elbows	Defines elbow-joint objects on the link.
Expression	Defines the expression.
ExtAttr	The element that defines iProcess extended attributes.
ExtendedDescription	Defines the second step description.
FieldDetails	Defines the original field type/length /format specification.
FieldMark	Defines the field marking.
FieldRef	Defines the unique ID of a field reference.
FieldType	Defines the field type.
FormDef	Defines complete form layout definition.
GraftStepFlags	Defines graft step property flags.
InputMappings	Defines input parameter mappings.
Label	Defines any labels.

Extended Attribute	Description
LayoutOptions	Defines the default layout/link style options.
Link	Defines the workflow process links that have no XPDL equivalent (see WithdrawStepLink for more information).
LinkStyleFlags	Defines extra link style flags.
LinkFlags	Defines extra link connection properties.
NonFlowObject	Defines procedure objects not involved in the process (see non-workflow objects for more information).
OldSubProcParam	Defines the sub-procedure field mappings.
OutputMappings	Defines output parameter mappings.
PredictDuration	Defines predicted duration.
ProcProperties	Defines procedure properties.
PublicStep	Defines any public steps.
ReturnSubProcName	Defines the array data field for return of grafted sub-procedure names.
RoleName	Defines the roles.
SchemaFormat	Defines the version of the iProcess Extended Attributes schema you are using. If it is not present when the XPDL document is loaded then it is assumed to be compatible with the schema format currently employed by the loading software.
Script	Defines any scripts.
ScriptRef	Defines the unique ID of a script.

Extended Attribute	Description
StartStepArray	Defines date field for sub-procedure start steps.
StepCommands	Defines the initial/keep/release command expressions.
StepFlags	Defines the step property flags.
StepPriority	Defines the work item priority settings.
SubProcCallParams	Defines the details of the sub-procedure I/O parameter mappings.
SubProcedureDetails	Defines details of the sub-procedure to call.
SubProcNameArray	Defines data fields for sub-procedure names.
SubPStepFlags	Defines step property flags.
SwimLanes	Defines the details of the swim lanes.
Template	Defines reference to the sub-procedure I/O parameter template procedure used by this object.
TextPosition	Defines the annotation label position.
UserName	Defines the name of the user.
XPDLExtAttr_XSD_Version	Defines the version of the iProcess Extended Attributes XSD.

Note:

- You can define some or all of the extended attributes, depending on your requirements. For example, you could specify the **LinkStyleFlags** extended attribute without specifying the **LinkFlags** extended attribute.
- Some extended attributes are ignored in the absence of another extended attribute. For example, the **TextPosition** attribute is ignored if the **ObjType** extended attribute is not present.

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the TIBCO Product Documentation website, mainly in HTML and PDF formats.

The TIBCO Product Documentation website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

The following documentation for this product is available on the TIBCO iProcess® Workspace (Windows) Product Documentation page:

- *TIBCO iProcess® Workspace (Windows) Release Notes*

  Read the release notes for a list of new and changed features. This document also contains lists of known issues and closed issues for this release.

- *TIBCO iProcess® Workspace (Windows) Installation*

  Read this manual for instructions on site preparation and installation.

- *TIBCO iProcess Suite Documentation Library*

  This library contains all the manuals for TIBCO iProcessWorkspace (Windows), TIBCO iProcess® Modeler, and other TIBCO products in TIBCO iProcess Suite. The manuals for TIBCO iProcess® Modeler and TIBCO iProcess® Modeler are the following:

  - *TIBCO iProcess Workspace (Windows) User Guide*
  - *TIBCO iProcess Modeler Getting Started*
  - *TIBCO iProcess Modeler Procedure Management*
  - *TIBCO iProcess Modeler Basic Design*
  - *TIBCO iProcess Modeler Advanced Design*

- *TIBCO iProcess Modeler Integration Techniques*
- *TIBCO iProcess Expressions and Functions Reference Guide*
- *TIBCO iProcess Workspace (Windows) Manager's Guide*
- *TIBCO iProcess COM Plug-in User Guide*
- *TIBCO iProcess Database Plug-in User Guide*
- *TIBCO iProcess Email Plug-in User Guide*
- *TIBCO iProcess Script Plug-in User Guide*
- *TIBCO iProcess Plug-in SDK User Guide*

Other TIBCO Product Documentation

When working with TIBCO iProcess® Modeler, you may find it useful to read the documentation of the following TIBCO products:

- TIBCO ActiveMatrix BusinessWorks™
- TIBCO Business Studio™
- TIBCO Enterprise Message Service™
- TIBCO Hawk®
- TIBCO Rendezvous®

How to Contact TIBCO Support

Get an overview of TIBCO Support. You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about products you are interested in, visit the TIBCO Support website.
- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to TIBCO Support website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community

offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the TIBCO Ideas Portal. For a free registration, go to TIBCO Community.

Legal and Third-Party Notices