TIBCO iProcess® Plug-in SDK

User Guide

Version 11.9.0
August 2022

Contents

Introduction to the TIBCO iProcess Plug-in SDK

This section provides an overview of the iProcess plug-in architecture, the system requirements you need for developing EAI plug-ins, what is included in the TIBCO iProcess Plug-in SDK and how to install the files onto your development machine.

Overview of the EAI Plug-in Architecture

For the EAI plug-in software to work with iProcess, it needs to consist of a server plug-in and a client plug-in. These two components perform the following tasks:

- The server plug-in is installed on the server and processes the step information.
- The client plug-in is installed on the TIBCO iProcess™ Workspace (Windows) using the TIBCO iProcess Modeler. It is used to provide a custom interface for defining the EAI plug-in definition data. This data is the information that is processed by the server plug-in such as populating some fields or retrieving some data from external systems. If you do not want to create your own client plug-in, you can use the Generic client plug-in that is installed by default with the TIBCO iProcess Modeler.

Note: The server plug-in and client plug-in are often referred to as the run-time plug-in and define-time plug-in respectively in the TIBCO iProcess Plug-in SDK source code.

Because of this architecture, the TIBCO iProcess Plug-in SDK is divided into two parts:

- EAI Server Plug-in SDK
- EAI Client Plug-in SDK

Each SDK contains all of the source code and examples that you need to build custom EAI plug-ins that can perform whatever tasks you require as part of an iProcess procedure (via an EAI step).

EAI Interface Versions

You can, if you wish, create EAI plug-ins that support multiple EAI interface versions and hence support older versions of the iProcess server.

As new versions of the EAI interface are developed, you need to make sure that your plug-in and the TIBCO iProcess Engine support the appropriate version(s) for your intended use. This makes sure that the appropriate functions will be used and that you implement the correct functions for the version of the interface you intend to support.

For example, to use the EAIRun_Withdraw_v2() function, your plug-in needs to support Version 2 of the interface specification and the TIBCO iProcess Engine also needs to support Version 2.

If the server only supports Version 1 but your plug-in supports Version 2, the server background will not be able to call EAIRun_Withdraw_v2(). However, if your plug-in supports Version 1 and 2 and you implement both versions of the withdraw function (EAIRun_Withdraw() and EAIRun_Withdraw_v2()), older servers that do not support V2 of the interface will call EAIRun_Withdraw() and newer servers will call EAIRun_Withdraw_v2().

For more information about defining the interface version(s) supported in your plug-ins, refer to EAIGWD_GetIFCVers() for client plug-ins and EAIRun_GetIFCVers() for server plug-ins.

System Requirements

To use the TIBCO iProcess Plug-in SDK to build Windows or UNIX EAI plug-ins, you require the following hardware and software:

Windows platform:

- Windows 10
- Microsoft Visual Studio 2017

UNIX platform:

- UNIX server
- UNIX C development tools

- Linux 6.6, Solaris 11 - GCC 6.3
- AIX 7.2 - GCC 5.2

The TIBCO iProcess Plug-in SDK can be used to develop EAI plug-ins for the TIBCO iProcess Engine.

Installing the TIBCO iProcess Plug-in SDK Files

The TIBCO iProcess Plug-in SDK is installed automatically as part of the TIBCO iProcess Engine installation. The files are placed in `SWDIR\eaisdk`.

Description of the TIBCO iProcess Plug-in SDK Source Source Files

The following tables describe the source files that are provided in the TIBCO iProcess Plug-in SDK.

EAI Server Plug-in SDK Files (UNIX)

The following source files are provided for building the EAI server plug-in on UNIX. These files can be found in the `eai_runtime_plug-in\unix` folder on the distribution CD.

Filename	Description
setup.unix	Environment script used to set up the correct environment variables on your UNIX platform.
makefile.unix	Makefile for building your custom plug-in. This also builds a distribution/installation set for your plug-in.
eairun.h	Header file that stores the types and definitions that are shared between the server background and the EAI plug-in.

Filename	Description
eaigwd.h	Header file that stores the types and definitions that are shared between the TIBCO iProcess Modeler and the EAI client plug-in.
eai_unx_install	Example script to install or upgrade a server plug-in.
eai_runtime_ shell.c	Template source code for the server plug-in. You can add specific code in here to determine what your EAI plug-in will do.

EAI Server Plug-in SDK Files (Windows)

The following source files are provided for building the EAI server plug-in on Windows. These files can be found in the `eai_runtime_plug-in\nt` folder on the distribution CD:

Filename	Description
eai_runtime_shell.c	Template source code for the server plug-in. You can add specific code in here to determine what your EAI plug-in will do.
eai_runtime_shell_ win32.dsw, .dsp, .ncb, .opt, .plg	Visual C++ project files.
eairun.h	Header file that stores the types and definitions that are shared between the server background and the EAI plug-in.
eaishell_run.def	Library file.

EAI Client Plug-in SDK Files

The following source files are provided for building the EAI client plug-in on Windows. These files can be found in the `EAI_SPD_Plugin` folder on the distribution CD:

Filename	Description
eairun.h	Header file that stores the types and definitions that are shared between the server background and the EAI plug-in.
eaigwd.h	Source header module that defines the types and definitions that are shared between the TIBCO iProcess Modeler and EAI plug-in library source.
eai_spd_shell_rc.def	Library file for resources.
eai_spd_shell_rc.c	Language resource code.
eai_spd_shell.def	Library file.
eai_spd_shell.c	Template source code for the server plug-in. You can add specific code in here to determine what your EAI plug-in will do.
EAI SPD Plugin.dsw, .plg, .opt, .ncb	Visual C++ project workspace files for client plug-in.
EAI SPD Plugin Resources.dsw, .plg, .opt, .ncb	Visual C++ project workspace files for client plug-in resources.

Creating an EAI Server Plug-in (UNIX)

This section explains how to create the EAI server plug-in component on UNIX platforms. You can create an EAI server plug-in with a specific EAI name and register it with the TIBCO iProcess Engine.

The EAI server plug-in needs to be coded to process the data that is supplied by the EAI client definition. For example, this can be some XML data that includes iProcess fields as tokens that are replaced at run-time. In this case, the server plug-in would parse the XML data and perform the required operations (such as replacing field tokens with field values).

Overview of Creating an EAI Server Plug-in

The `eai_runtime_shell.c` source file provides the source code for a shell EAI server plug-in. This contains all of the required interface functions and generic code performed by the plug-in. The functions are empty and are coded to return a success value. You can add your own code into these functions as required.

Note: TIBCO recommend that you perform a test build and installation of the source code before you start adding your own code. This ensures that your environment is correctly set up to build the plug-in.

The following is a brief overview of the steps you need to perform to create your EAI server plug-in:

- Define the EAI type name for your plug-in and set up the UNIX environment variables - see Defining the EAI Plug-in Type Name and Environment Setup.
- Build the server plug-in - see Building the EAI Server Plug-in.
- Install the server plug-in - see Installing Your EAI Server Plug-in.
- Edit the plug-in source code to create your custom plug-in - see Customizing the EAI Run-Time Plug-in.

Defining the EAI Plug-in Type Name and Environment Setup

The EAI type name is used to identify what type of EAI plug-in you are creating.

To define your EAI type name, do the following:

1. Edit the EAITYPE environment variable in the environment set-up script `setup.unix`. Change EAI_SHELL on the following line to your chosen EAI type name:

   ```
   export EAITYPE=EAI_SHELL
   ```

   The type name can be up to 20 alpha numeric characters but must not contain spaces or tabs.

> **Note:** This name must be the same as the client name defined in your client plug-in - see Overview of Creating an EAI Server Plug-in. However, you can customize the client EAI plug-in to use a server plug-in with a different name - see EAIGWD_GetInfo() for more information.

2. Edit the release version of your plug-in by changing the following line in the `eai_runtime_shell.c` file to your required version number:

   ```
       #define RELEASE_VERSION “1.0”
   ```

   — name the installed plug-in (all non-numerics are replaced by _).

   — display the upgrade prompt when `SWDIR\util\sweaireg` is used.

3. Set up the build environment by running the `setup.unix` script. This script is designed to autodetect your operating system and set up the build environment accordingly (such as the compiler and linker tools to use and flags to pass to them). Change to your server plug-in source directory and enter the following command:

   ```
       . ./setup.unix
   ```

   The leading . (period+space) must be entered so that the environment settings changed by `setup.unix` are inherited by your environment.

Note: You must run this script **once** per login or if you change the EAI type name before performing a build.

Building the EAI Server Plug-in

To build your EAI server plug-in, go to the server plug-in directory in which you have the server plug-in source files and enter the following command:

```
make -f makefile.unix
```

This compiles the `eai_runtime_shell.c` file and links it into a shared library called *eaitype*`.ear` where *eaitype* is the EAI type name you defined in the `setup.unix` script. The `.ear` file is your EAI server plug-in.

This command also builds the EAI server plug-in installation set for your plug-in in a new `installset` sub-directory. This contains the `.ear` file and an installation script for it called *eaitype*`_install` where *eaitype* is the EAI plug-in type name. This install script is a copy of the `eai_unx_install` script file that has been pre-processed to use your EAI type name.

Installing Your EAI Server Plug-in

After successfully building your EAI server plug-in, you now have to install it on your TIBCO iProcess Engine. To do this:

1. Log on to the TIBCO iProcess Engine as the iProcess background user or, on UNIX, as the super-user.
2. In the `installset` directory, enter the following command

   ```
   ./EAITYPE_install
   ```

   where *EAITYPE* is your EAI type name (as defined in the `setup.unix` script file - see Defining the EAI Plug-in Type Name and Environment Setup.)
3. Follow the on-screen instructions.

   The install script registers the EAI plug-in with your TIBCO iProcess Engine and the installation is complete.

If you already have procedures defined with EAI steps of the same type, you can now run these procedures. If you need to create the client EAI plug-in refer to Creating an EAI Client Plug-in.

4. The server plug-in (`.ear` file) is copied to your chosen directory and renamed as EAITYPE_*release_version*`.ear` where *release_version* is the string defined in `eai_runtime_shell.c` on the **`#define RELEASE_VERSION`** line.

Customizing the EAI Run-Time Plug-in

After completing a test build and installation of your EAI server plug-in, you need to customize the functions provided in the `eai_runtime_shell.c` source file.

Warning: You must not change the parameter definitions to these functions or their return codes. Only use the return codes that are specified in the interface descriptions - see EAI Client Plug-in Interfaces.

Each function's header comment describes its parameters, the purpose of the function and when the TIBCO iProcess Engine calls the function. Refer to EAI Client Plug-in Interfaces for more information about the functions you can use.

Your run-time plug-in may need to be compiled and linked with other standard and non-standard libraries, build and link flags, headers and source, etc. You can edit the `makefile.unix` and `setup.unix` files to do this.

If your plug-in requires an initialization parameter string in the EAI server plug-in registry (a string stored in the iProcess database and passed to your plug-in by the TIBCO iProcess Engine) you can modify the call made to `sweaireg` in the `eai_unx_install` script as appropriate. For example, you can use an initialization parameter string to define the connection setting for an external application. When the plug-in loads, the initialization parameter is used to connect to the external application.

Creating an EAI Server Plug-in (Windows)

This section explains how to create the server plug-in component on Windows platforms. You can create an EAI server plug-in with a particular EAI type name and register it with the TIBCO iProcess Engine.

Overview of Creating an EAI Server Plug-in

The `eai_runtime_shell.c` source file provides source for a shell server plug-in. This contains the code for all of the required interface functions and generic code performed by the plug-ins. The functions are empty and are coded to return a success value. You can add your own code into these functions as required.

Note: TIBCO recommend that you perform a test build and installation of the source code before you start adding your own code. This ensures that your environment is correctly set up to build the plug-in.

The following is a brief overview of the steps you need to perform to create your EAI server plug-in:

- Build the server plug-in - see Building the EAI Server Plug-in.
- Install and register the server plug-in - see Installing and Registering the EAI Server Plug-in.
- Customize the shell source code to add in your specific plug-in functionality - see Customizing the EAI Run-Time Plug-in.

Building the EAI Server Plug-in

To build your EAI server plug-in, do the following in MS Visual C++ to build the `eaishell.ear` dll component:

1. Open the `eai_runtime_shell_win32.dsw` project.

Note: If you want to specify a different output filename instead of `eaishell.ear`, click **Project > Settings** and click the **Link** tab. Enter a new name in the **Output File Name** field and click **OK**.

2. Click **Build > Batch Build**. You can choose to build either the Release or Debug versions or choose both components.
3. Click **Rebuild All**.

   This compiles the `eai_runtime_shell.c` file into a DLL called `eaishell.ear` (or the output filename you specified in the project settings. The `.ear` file is your EAI server plug-in.

Installing and Registering the EAI Server Plug-in

You need to install and register your EAI server plug-in with the TIBCO iProcess Engine:

1. Log in to your TIBCO iProcess Engine as the background user (pro by default).
2. Copy the eaishell.ear file to the SWDIR\lib folder. This is the standard location for EAI server plug-ins but you can use a different location if you need to. The path is registered on the TIBCO iProcess Engine when you register the plug-in.
3. (Optional) You can set debug on the new plug-in so that you can trace any problems you have when registering or loading it. To set the debug level, enter the following command (from the SWDIR\util folder):

   ```
   swadm set_attribute 1 BG 0 DEBUG c=1,x=255
   ```

   This causes iProcess to output trace logging for EAI plug-ins to the `bg`*process_instance_number*`.log` file.
4. Register the plug-in by entering the following command (from the `SWDIR\util` folder):

   ```
   sweaireg REG eaishell -L SWDIR\lib\eaishell.ear
   ```

   where *eaishell* is the EAI step type name that the given `.ear` library handles. If you are using multiple iProcess servers in a node cluster, you need to repeat this step for each server.

Customizing the EAI Run-Time Plug-in

After completing a build and installation of your EAI server plug-in, you can now customize the functions provided in the `eai_runtime_shell.c` source file.

Warning: You must not change the parameter definitions to these functions or their return codes.

Each function's header comment describes its parameters, the purpose of the function and when the iProcess background calls the function. Refer to EAI Client Plug-in Interfaces for more information about the interface specification.

Creating an EAI Client Plug-in

This section describes how to create an EAI client plug-in for the Windows platform. The client plug-in is used by the TIBCO iProcess Modeler to enable procedure definers to create and edit the definition of the EAI step. When a case of the procedure is run, the associated EAI server plug-in processes the step data.

If you don't want to create a specific EAI client plug-in, you can use the Generic EAI client plug-in that is installed by default with the TIBCO iProcess Modeler to manually edit the step definition for your EAI step type. Refer to Using the Generic EAI Client Plug-in for more information about using the generic plug-in.

Overview of Creating an EAI Client Plug-in

The following section provides an overview of the steps you need to perform to create a new EAI client plug-in:

1. In MS Visual C++, open the EAI client project workspace (`EAI SPD PlugIn.dsw`). This workspace contains two projects:
   - TIBCO iProcess Modeler Plug-in

     This project builds the `eai_spd_shell.eag` dynamic link library.
   - TIBCO iProcess Modeler Plug-in resources

     This project builds the `eai_spd_shell_rc.dll`, which is the language resource dynamic link library used by the plug-in at run-time. This project also builds the installation set for your plug-in.
2. Define the EAI type name for the plug-in. Open the `eai_spd_shell.c` file and change the value of *EAI_SHELL* on the following line to your specific EAI type name:

   ```
       #define EAI_TYPE_NAME “EAI_SHELL”
   ```

   You can also change the value of the *RELEASE_VERSION* definition to your new version number if you are creating an updated version of your plug-in.

Building the EAI Client Plug-in

To build your EAI client plug-in, do the following:

1. In MS Visual C++, click **Build > Batch Build**.
2. In the **Batch Build** dialog box, make sure that all the components are selected and click **Rebuild All**. The installation set for your plug-in is only created if you perform a build this way.
3. Check the Output window to check that the components have built correctly.

The following sub-directories are created in your working folder:

— `EAI_installSet` - contains the install set for the `.eag` client plug-in.

— `Release` - contains release version of `.eag` file and objects.

— `Debug` - contains debug version of `.eag` file and objects.

Installing the EAI Client Plug-in

To install the client plug-in on your TIBCO iProcess Workspace (Windows), do the following:

1. Log in to your TIBCO iProcess Workspace (Windows) and open the TIBCO iProcess Modeler.
2. Click **Options > Install EAI Plug-in**.
3. Locate the `EAI_InstallSet` folder, select the `eai_spd_shell.eag` file and click **Open**.
4. The EAI client plug-in is now installed. You can check which plug-ins you have installed by clicking **Help** > **About** and looking at the list of installed plug-ins. The EAI type name and version number are displayed in this list.

Customizing the EAI Client Plug-in

After confirming that the build and installation of your plug-in works, you can customize the EAI dialogs and their supporting procedure code.

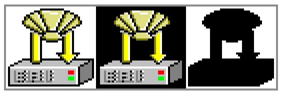
Before editing or creating the client dialogs, consider the following tasks that your dialog box code will need to perform:

- Interpret the current EAI step type definition data passed by the TIBCO iProcess Modeler into an appropriate data structure. This is the last data returned to the TIBCO iProcess Modeler by your plug-in for the given step.
- Transfer the step definition data into the controls in your dialog box.
- Extract the new data from the dialog box controls and create the step definition as a text buffer after the user clicks **OK**.

The text that is extracted from the dialog box is in a format of your choosing. It is the text that will be passed back to your TIBCO iProcess Modeler plug-in when the step is edited and to your run-time plug-in when the TIBCO iProcess Engine is processing cases containing this EAI step.

Editing the EAI Client Plug-in EAI Step Icon

The shell source code contains the following generic resource bitmap image for the EAI client plug-in:



You can create your own icon for your client plug-in by editing this bitmap or creating a new bitmap. Please note the following when creating your bitmap image:

- The left hand image is a full picture using a white background.
- The middle image is the same picture with all transparent parts set to absolute black (as well as the black parts).
- The right hand image is the same picture but is a negative. It must contain only pure white and absolute black. Use white for the sections you need to be transparent and black for the sections that are to overlay the background window color of the TIBCO iProcess Modeler.
- The three image sections must be of equal width and the bitmap must only contain these images (no borders or surplus space).

For more information about displaying the step icon, refer to EAIGWD_GetImage().

Using the Generic EAI Client Plug-in

If you do not want to create your own custom client plug-in, you can use the Generic EAI client plug-in to edit the EAI definition data. The plug-in provides two multi-line edit controls that enable the entry of the EAI type data for the process step and the withdraw step. However, you need to enter the information in the raw format specified for the plug-in.

Installing the Generic plug-in

The generic plug-in calculates the name of the EAI type from the library file it is installed as. Therefore, you can make a copy of the `eai_gen.eag` library file and rename it to your EAI type name to match your server plug-in. To do this:

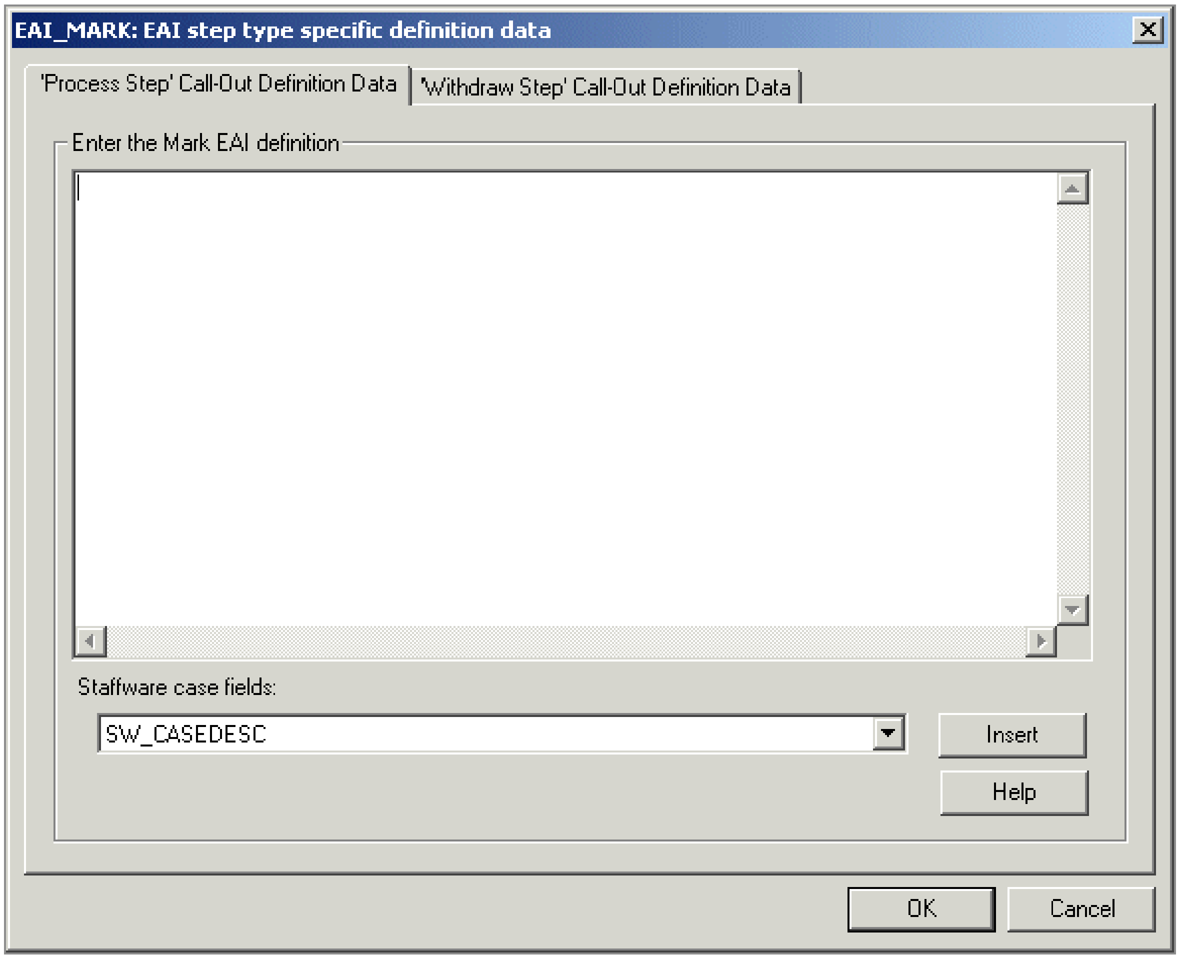
1. Create a new folder in the `eai_plugins` folder with the required EAI type name.
2. Copy the contents of the existing `eai_plugins\eai_gen` folder into your new folder.
3. Rename the `eai_gen.eag` file to *typename*`.eag`. For example, if you have created a server plug-in with a type name of *mytype*, your file will be called `mytype.eag`.
4. Start the TIBCO iProcess Workspace (Windows) and from the TIBCO iProcess Modeler install your new plug-in by clicking **Options** > **Install EAI Plug-in**.

Editing the EAI Step Definition Data

After installing the Generic EAI Plug-in, you can enter the EAI specific data in the **EAI step type specific definition data** dialog box. To do this:

1. In the TIBCO iProcess Modeler, create a new EAI step on your procedure.

   The EAI Step Properties pane is displayed.
2. Enter the name of the step and select the EAI Plug-in type from the **EAI Type** drop down list (by default this is **EAI_SHELL - EAI SDK Shell Plug-in**).
3. Click **Edit Call-Out Definition**. The following dialog box is displayed, which enables you to enter the required data for your EAI step.


EAI_MARK: EAI step type specific definition data
'Process Step' Call-Out Definition Data
'Withdraw Step' Call-Out Definition Data
Enter the Mark EAI definition
Staffware case fields:
SW_CASEDESC
Insert
Help
OK
Cancel

EAI Client Plug-in Interfaces

This section provides reference information for the available EAI client plug-in interfaces. The `eai_spd_shell.c` source file contains the shell functions and comments about what they do. The functions provided in this file are empty but they all return a success value.

After compiling the `eai_spd_shell.c` code, a dialog box is displayed at run-time that enables you to edit the source EAI type specific definition data for your own EAI step type. The dialog box and its window procedure code can be changed as desired to create a user interface that meets the step definition requirements that are unique to your EAI Type. For example, if you just need three pieces of data for your EAI step, you can design a dialog box that provides 3 fields for each of these pieces of data.

The following public functions are described first:

- EAIGWD_EditStep()
- EAIGWD_DisplayHelp()
- EAIGWD_GetInfo()
- EAIGWD_GetIFCVers()
- EAIGWD_Init()
- EAIGWD_Term()
- EAIGWD_GetImage().

The following functions are not defined by the plug-in but are provided by the TIBCO iProcess Modeler and can be called by the plug-in:

- gwdEAI_GetCallBack()
- gwdEAI_GetFields()
- gwdEAI_ReAlloc()
- gwdEAI_GetStepInfo()
- gwdEAI_CheckExpr()
- gwdEAI_GetFields_Ex().

The EAI GWD plug-in interface types and definitions are stored in the `eaigwd.h` source header file.

Note: Due to the name of the Graphical Workflow Definer (GWD) changing to the TIBCO iProcess Modeler, the code functions and comments still refer to the GWD.

EAIGWD_EditStep()

Synopsis

```
int EAIGWD_EditStep (
   void                *PlugInContext,
   void                *GWDContext,
   HWND                *hWndParent,
   char               **lplpRunSectData,
   long                *lpRunSectSize,
   char                *lplpWithdrawSectSize,
   long                *lpWithdrawSectSize,
   char                *lplpGWDSectData,
   long                *lpGWDSectSize)
```

Parameters

Name	Description
PlugInContext	The value returned by the plug-in for a previous call to EAIGWD_Init().
GWDContext	The context handle of the TIBCO iProcess Modeler. This is required by certain call back functions accessible to the plug-in during the **EAIGWD_EditStep()** call.
hWndParent	The handle of the TIBCO iProcess Modeler window (parent window) of any dialog box box displayed by the plug-in.

Name	Description
lplpRunSectData	A pointer to the address of the currently defined data for the process step data as a NULL terminated string. This data buffer is dynamically allocated by the TIBCO iProcess Modeler and should be overwritten by this function on success with the new definition data for the section. On success, if the new definition data for the section is larger than the current size (*lpRunSectSize) it should be reallocated using the gwdEAI_ReAlloc() call back function.
lpRunSectSize	The current size of the buffer addressed by the pointer value *lplpRunSectData. This is the entire extent of the buffer and not just the currently used extent.
lplpWithdrawSectData	This is the same as **lplpRunSectData** but is for the withdraw step data.
lpWithdrawSectSize	This is the same as **lpRunSectSize** but is used for the **lplpWithdrawSectData** section.
lplpGWDSectData	This is the same as **lplpRunSectData** but is used for the edit time only data.
lpGWDSectSize	This is the same as **lpRunSectSize** but is used for the **lplpGWDSectData** section.

Description

This function is called when the user clicks the Edit EAI type specific data button in the common Properties pane. In most cases, the plug-in will display a dialog box that enables the user to edit the data for the process step and withdraw step sections. Whatever is coded must be modal, which means that the TIBCO iProcess Modeler expects the editing work to be complete when this function returns.

Returns

Value	Description
EAIGWD_ COMPLETE	Success. The EAI type specific definition data is complete. This means that everything required by the run-time plug-in is defined.
EAIGWD_ INCOMPLETE	Success. However, the EAI type is partially defined so the TIBCO iProcess Modeler displays the step as incomplete and the step is saved with a status of incomplete.
EAIGWD_ USERCANCEL	The user cancelled the editing dialog box.
EAIGWD_ERROR	Error. The TIBCO iProcess Modeler attempts to preserve the current section data for the step.

EAIGWD_DisplayHelp()

Synopsis

```
void EAIGWD_DisplayHelp (
   void                 *PlugInContext,
   HWND                  hWndParent)
```

Parameters

Name	Description
PlugInContext	The value returned by the plug-for a previous call to EAIGWD_Init ().
hWndParent	The TIBCO iProcess Modeler window handle.

Description

This function is optional. You can use it to add an extra item to the TIBCO iProcess Modeler **Help** menu for the plug-in’s online help. When the user selects this new menu item, the TIBCO iProcess Modeler calls this function so the plug-in can display the online help. The system used for displaying the online help is the responsibility of the plug-in.

Returns

None.

EAIGWD_GetInfo()

Synopsis

```
    int EAIGWD_GetInfo (
       int               InfoID,
       char             *lpInfoRet,
       int               cbInfoRet)
```

Parameters

Name	Description
InfoID	Identifies what data is required for return.
lpInfoRet	A pointer used to return the data string buffer.
cbInfoRet	The maximum number of bytes that can be written to **lpInfoRet** (including the NULL terminator).

Description

This function is used to get information about the plug-in. This function should return the information indicated by the **InfoID** parameter as a NULL terminated string in the **lpInfoRet** parameter.

InfoID can be one of the following values:

InfoID	lpInfoRet Return Value
EAIGWD_ INFO_ EAITYPE	EAI type name supported by this plug-in (maximum of 20 characters).
EAIGWD_ INFO_ RELVERS	The release version of the plug-in (maximum of 32 characters).
EAIGWD_ INFO_DESC	Text description of the EAI step type information (maximum of 32 characters).
EAIGWD_ INFO_ BTNTEXT	The text that the TIBCO iProcess Modeler displays on the Edit EAI type specific data button.
EAIGWD_ INFO_ SUPPORT_ DELREL	Returns **lpinfoRet** if passed as NULL and should not be assigned. The function return code should be EAIGWD_DELRELOK if the server plug-in supports delayed release or EAIGWD_NODELREL if it does not support delayed release. In this case the TIBCO iProcess Modeler disables all the controls in the **Delayed Release** tab of the EAI step Properties pane. If neither of these return codes are used, the default is EAIGWD_DELRELOK.
EAIGWD_ INFO_ RUNTIME	Defines the EAI type name of the server plug-in that is supported by your client plug-in (maximum of 20 characters). If EAI_NOTSUPPORTED is returned for this **InfoID**, the EAI type name of your client plug-in is used as the default server plug-in type name. For example, if your client plug-in is called EAI_JAVA but you want it to use a server plug-in called EAI_JAVAR1, set EAIGWD_INFO_RUNTIME to EAI_JAVAR1.

Returns

Value	Description
EAIGWD_OK	Success.
EAIGWD_ NOTSUPPORTED	The plug-in is not coded to return a value for the requested **InfoID**. You can return this if you do not recognize the passed **InfoID** value (i.e. it is not one of the listed values).
EAIGWD_DELRELOK	Return this value if passed an **InfoID** parameter value of **EAIGWD_ INFO_SUPPORT_DELREL** and the server plug-in supports delayed release.
EAIGWD_ NODELREL	Return this value if passed an **InfoID** parameter value of **EAIGWD_ INFO_SUPPORT_DELREL** and the server plug-in does not support delayed release.

EAIGWD_GetIFCVers()

Synopsis

```
    int const *EAIGWD_GetIFCVers (void)
```

Parameters

None.

Description

This function is used to determine the EAI TIBCO iProcess Modeler plug-in interface specification version supported by the plug-in. This version is determined by iProcess with each release of the TIBCO iProcess Plug-in SDK and can be found in the source code. This function must return a list of the interface specification version numbers of the interface versions supported by the plug-in.

The version number list must be a zero terminated array of integer values in descending order. For example, if the plug-in supports version 1 of the interface, this array is declared as:

```
static int      fc_versions[]={1,0};
```

For a plug-in that supports interfaces specification version 1, 2 and 3, the array is declared as:

```
static int   ifc_versions[]={3,2,1,0};
```

Note: The descending order is used so that the TIBCO iProcess Engine chooses the highest specification version supported by it and the plug-in. You can also omit version numbers as long as the descending order is preserved. For example {4,1,0} indicates that version 4 and 1 is supported but not 2 and 3.

Returns

Value	Description
!NULL	Pointer to version integer array.

EAIGWD_Init()

Synopsis

```
void *EAIGWD_Init (
   int              IFCVers,
                    GWDEAI_GETCALLBACK_FN
                    gwdEAI_GetCallBack)
```

Parameters

Name	Description
IFCVers	Indicates the EAI TIBCO iProcess Modeler plug-in interface specification version that the TIBCO iProcess Modeler will use from the list returned by the EAIGWD_GetIFCVers().
gwdEAI_ GetCallBack	The address of the "get call-back" function in the TIBCO iProcess Modeler. This can be used by the plug-in to load other TIBCO iProcess Modeler EAI call-back functions - see gwdEAI_GetCallBack() for more information.

Description

This function is called every time the plug-in library is loaded after the TIBCO iProcess Modeler determines which interface version number is supported for the TIBCO iProcess Modeler and the plug-in.

Returns

Value	Description
NOT NULL	Success. A context identifier passed to other **EAIGWD_** functions.
NULL	Error.

EAIGWD_Term()

Synopsis

```
void *EAIGWD_Term (
   void              PlugInContext)
```

Parameters

Name	Description
PlugInContext	The value returned by the plug-in for a previous call to EAIGWD_ Init(). This can be passed as NULL where the plug-in has been loaded by the TIBCO iProcess Modeler but the **EAIGWD_Init()** function has not been called before the TIBCO iProcess Modeler then unloads the plug-in. This enables the plug-in to free any resources that may be allocated before **EAIGWD_Init()** was called.

Description

This function is called by the TIBCO iProcess Modeler when it needs to unload the plug-in. This enables the plug-in to free any resources allocated and saved during the instance of the plug-in.

Returns

None.

EAIGWD_GetImage()

Synopsis

```
    HBITMAP EAIGWD_GetImage (
       void                  *PlugInContext)
```

Parameters

Name	Description
PlugInContext	The value returned by the plug-in for a previous call to EAIGWD_Init().

Description

This function should return the Windows bitmap image handle that is used to display EAI step objects of the type supported by the plug-in. This handle is usually loaded using the Windows **LoadBitmap()** function for a bitmap resource built into the plug-in itself.

If this function returns zero, the EAI step objects in the workflow of the type this plug-in supports are displayed with the default EAI step object image. Refer to Editing the EAI Client Plug-in EAI Step Icon for more information about customizing the default step object.

Returns

Value	Description
NOT 0	Windows bitmap handle
0	No image defined so the default is used.

gwdEAI_GetCallBack()

Note: This function is not defined by the plug-in. It is provided by the TIBCO iProcess Modeler code at run-time and can be called by the plug-in.

Synopsis

```
void *gwdEAI_GetCallBack (int fnID)
```

Parameters

Name	Description
fnID	The call back function identifier that determines the call back function pointer to return.

Description

This function returns a pointer to various call-back functions that are required by the EAI plug-in. The address of this function is passed to the EAIGWD_Init() function when the TIBCO iProcess Modeler initializes the plug-in. The following table describes the call back functions that can be accessed via this function, the value of the **fnID** parameter to use and the type definition that this function returns for each.

Call Back Function	fnID	Return Type
gwdEAI_GetFields()	GWDEAI_ GETFIELDS	GWDEAI_GETFIELDS_FN
gwdEAI_ReAlloc()	GWDEAI_ REALLOC	GWDEAI_REALLOC_FN
gwdEAI_GetStepInfo()	GWDEAI_ GETSTEPINFO	GWDEAI_GETSTEPINFO_FN
gwdEAI_CheckExpr()	GWDEAI_ CHECKEXPR	GWDEAI_CHECKEXPR_FN

Returns

Value	Description
NOT NULL	The address of the call-back function identified by fnID.
NULL	Invalid fnID value.

gwdEAI_GetFields()

Note: This function is not defined by the plug-in. It is provided by the TIBCO iProcess Modeler code at run-time and can be called by the plug-in.

Synopsis

```
char gwdEAI_GetFields (
   void                 *GWDContext,
   int                   bFirst,
   char                 *lpFldName,
   int                  *lpFldLen,
   int                  *lpFldDpMid)
```

Parameters

Name	Description
GWDContext	The context handle for the TIBCO iProcess Modeler as passed to EAIGWD_EditStep().
bFirst	If non-zero, the first field definition is used. If zero, the next field definition is used
lpFldName	A pointer to the memory space used for the return of the field name

Name	Description
	(must be at least 32 bytes in length).
lpFldLen	The length of the field in bytes not including the NULL terminator: • This is the defined maximum length of the field and not the length of its current value. • This is not used for MEMO fields. • For numeric fields with decimal places, this includes the decimal places and the decimal point. For example, a field whose format is 9999.99 will have a length of 7 with 2 decimal places. • If this value is not required, this parameter can be passed as NULL.
lpFldDpMid	For numeric fields, this is the number of decimal places. For MEMO fields, this is the memo field identifier. If this value is not required, it can be passed as NULL.

Description

This function is used to get the first or next field definition in the list of iProcess case fields that are made available to the server plug-in. This function only lists fields that are supported by the TIBCO iProcess Engine at run-time.

Returns

Value	Description
NOT 0	Field type. This is one of the **EAI_FTYP_** values that are defined in `eairun.h`.
0	No more case fields to process.

gwdEAI_ReAlloc()

Note: This function is not defined by the plug-in. It is provided by the TIBCO iProcess Modeler code at run-time and can be called by the plug-in.

Synopsis

```
int gwdEAI_ReAlloc (
   char             **lplpSectData,
   long              *lpCurrSectSize,
   long               NewSectSize)
```

Parameters

Name	Description
lplpSectData	Used as a pointer to a pointer for the section data memory block as passed to EAIGWD_EditStep().
lpCurrSectSize	A pointer to the current section data buffer size.
NewSectSize	The new size for the section data memory block.

Description

This function is used to re-allocate memory that was originally allocated by the TIBCO iProcess Modeler. An EAI plug-in must use this function instead of calling the system **realloc()** function directly on any of the EAI section data buffers that are passed to its EAIGWD_EditStep() function.

Returns

Value	Description
NOT 0	Success. The contents of **lplpSectData** are updated to the pointer to the new memory block and the contents of **lpCurrSectSize** are updated to the new size.
0	Failure. The original memory remains untouched and the **lplpSectData** and **lpCurrSectSize** are not changed.

gwdEAI_GetStepInfo()

Note: This function is not defined by the plug-in. It is provided by the TIBCO iProcess Modeler code at run-time and can be called by the plug-in.

Synopsis

```
void gwdEAI_GetStepInfo (
   void                   *GWDContext,
   char                    lpStepName,
   int                     cbMaxName,
   char                   *lpStepDesc,
   int                     cbMaxDesc)
```

Parameters

Name	Description
GWDContext	The context handle for the TIBCO iProcess Modeler as passed to EAIGWD_EditStep().
lpStepName	A pointer to the memory space for the return of the step name. This

Name	Description
	is ignored if it is passed as a NULL pointer.
cbMaxName	The maximum number of bytes that can be assigned **lpStepName** including the NULL terminator.
lpStepDesc	A pointer to the memory space for the return of the step description. This is ignored if it is passed as a NULL pointer.
cbMaxDesc	The maximum number of bytes that can be assigned to **lpStepDesc** including the NULL terminator.

Description

This function enables the plug-in to access the step name and description of the EAI step being edited by the EAIGWD_EditStep() function.

Returns

Value	Description
void	The step name and description is returned via the **lpStepName** and **lpStepDesc** parameters.

gwdEAI_CheckExpr()

Note: This function is not defined by the plug-in. It is provided by the TIBCO iProcess Modeler code at run-time and can be called by the plug-in.

Synopsis

```
    int gwdEAI_CheckExpr (
        void                    *GWDContext,
    char                *lpExpression)
```

Parameters

Name	Description
GWDContext	The context handle for the TIBCO iProcess Modeler as passed to EAIGWD_EditStep().
lpExpression	A pointer to the NULL terminated expression string.

Description

This call-back function is used to validate the given iProcess expression string and returns the return type of valid expressions.

Returns

Value	Description
EAI_EXPR_BOOL	**lpExpression** is a valid BOOLEAN expression.
EAI_EXPR_TEXT	Returns a textual value.
EAI_EXPR_NUMERIC	Returns a numeric value.
EAI_EXPR_DATE	Returns a date value.
EAI_EXPR_TIME	Returns a time value.
EAI_EXPR_VAL_ TOOBIG	The expression string is too long. The maximum is currently 250 characters).
EAI_EXPR_INVALID	The expression is invalid.
EAI_ERROR	The parameter is invalid or there is another system error.

gwdEAI_GetFields_Ex()

Note: This function is not defined by the plug-in. It is provided by the TIBCO iProcess Modeler code at run-time and can be called by the plug-in.

Synopsis

```
char gwdEAI_GetFields_Ex (
   void                 *GWDContext,
   int                   bFirst,
   char                 *lpFldName,
   int                  *lpFldLen,
   int                  *lpFldDpMid,
   int                  *lpFldFlags))
```

Parameters

Name	Description
GWDContext	The context handle for the TIBCO iProcess Modeler as passed to EAIGWD_EditStep().
bFirst	If non-zero, the first field definition is used. If zero, the next field definition is used
lpFldName	A pointer to the memory space used for the return of the field name (must be at least 32 bytes in length).
lpFldLen	The length of the field in bytes not including the NULL terminator: • This is the defined maximum length of the field and not the length of its current value. • This is not used for MEMO fields. • For numeric fields with decimal places, this includes the decimal places and the decimal point. For example, a field whose format is 9999.99 will have a length of 7 with 2 decimal

Name	Description
	places. • If this value is not required, this parameter can be passed as NULL.
lpFldDpMid	For numeric fields, this is the number of decimal places. For MEMO fields, this is the memo field identifier. If this value is not required, it can be passed as NULL.
lpFldFlags	A pointer to the memory space for the return of a value that determines whether the field is an array field. If the EAI_FFLAG_ARRAY (value 0x0010) bit in the returned value is set, the field is an array field. If it is not set, the field is not an array field.

Description

This function is used to get the first or next field definition in the list of iProcess case fields that are made available to the server plug-in and determine whether or not it is an array field.

This function only lists fields that are supported by the TIBCO iProcess Engine at run-time.

Returns

Value	Description
NOT 0	Field type. This is one of the **EAI_FTYP_** values that are defined in `eairun.h`.
0	No more case fields to process.

EAI Server Plug-in Interfaces

This section provides reference information for the available EAI server plug-in interfaces. The `eai_run_shell.c` source file contains the shell functions and comments about what they do. The functions provided in this file are empty but they all return a success value.

The following public functions are described first:

- EAIRun_GetIFCVers()
- EAIRun_Init()
- EAIRun_Term()
- EAIRun_Call()
- EAIRun_Getreply()
- EAIRun_Withdraw()
- EAIRun_Withdraw_v2()
- EAIRun_GetLastError()
- EAIRun_CallCleanUp()
- EAIRun_GetReleaseVers().

The following functions are not defined by the plug-in but are provided by the TIBCO iProcess Modeler and can be called by the plug-in:

- proEAI_GetCallBack()
- proEAI_Debug()
- proEAI_DescribeData()
- proEAI_GetData()
- proEAI_FreeData()
- proEAI_SetData()
- proEAI_EvaluateExpr()
- proEAI_GetDateLine()
- proEAI_GetDataEx()

- proEAI_DescDataEx().

The EAI server plug-in interface types and definitions are stored in the `eairun.h` source header file.

EAIRun_GetLastError()

Synopsis

```
int EAIRun_GetLastError (
   char                      *lpErrorText,
   int                        cbErrorText)
```

Parameters

Name	Description
lpErrorText	Text description of the last error that occurred when the background process called any of the **EAIRun_**() functions in the plug-in.
cbErrorText	The maximum number of bytes that this function is allowed to write to the **lpErrorText** parameter, including the NULL terminator.

Description

The background process calls this function after a call to one of the other **EAIRun_** () functions has returned an error.

The background process typically uses the error text as part of an SW_WARN message it logs because of the failure.

Returns

Value	Description
Integer	Plug-in specific error code.

EAIRun_GetIFCVers()

Synopsis

```
const int *EAIRun_GetIFCVers (void)
```

Parameters

None.

Description

This function is used to return a list of the EAI server plug-in interface specification version numbers of the interface version supported by the plug-in. The interface specification is determined by iProcess with each release of the TIBCO iProcess Plug-in SDK.

The version number list must be a zero terminated array of integer values in descending order. For example, if the plug-in supports version 1 of the interface, this array is declared as:

```
static int   ifc_versions[]={1,0};
```

For a plug-in that supports interfaces specification version 1, 2 and 3, the array is declared as:

```
static int   ifc_versions[]={3,2,1,0};
```

Note: The descending order is used so that the TIBCO iProcess Engine chooses the highest specification version supported by it and the plug-in. You can also omit version numbers as long as the descending order is preserved. For example {4,1,0} indicates that version 4 and 1 is supported but not 2 and 3.

Returns

Value	Description
!NULL	Pointer to version integer array.

EAIRun_Init()

Synopsis

```
int *EAIRun_Init (
   int                          IFCVers,
   const char                  *lpInitParams,
   PROEAI_GETCALLBACK_FN        proEAI_GetCallBack)
```

Parameters

Name	Description
IFCVers	Contains the interface specification version number that the TIBCO iProcess Engine will use. This is obtained after calculating the highest version supported by the server and the server plug-in via the EAIRun_GetIFCVers().
lpInitParams	This is the NULL terminated string used for the initialization parameters defined in the EAI run-time registry for this plug-in. This can be used to define start-up parameters that are used by this

Name	Description
	function to tell the plug-in how to start-up. For example, this could be the connection details for an external application that the plug-in communicates with.
proEAI_ GETCallBack	Provides a way of accessing the other call back functions available to the plug-in.

Description

This function can be used to perform the initialization of any information that will last the duration of the EAI server plug-in process such as connection details for external systems.This function is called by the background process when:

- The background starts.
- The plug-in is installed or upgraded while the background is still running.
- The registry entry for the EAI plug-in is modified (such as adding initialization parameters).

Returns

Value	Description
EAI_ SUCCESS	Success.
EAI_ERROR	Failure. Processing of EAI steps by this plug-in will fail until the problem is fixed and the plug-in is reloaded using the `SWDIR\util\sweaireg` RELOAD command.

EAIRun_Term()

Synopsis

```
    void *EAIRun_Term (void)
```

Parameters

None.

Description

This function is called by the iProcess background process when it needs to stop access to the plug-in. This can be when the background process shuts down or when the plug-in is reloaded and re-initialized with the TIBCO iProcess Engine.

Returns

None.

EAIRun_Call()

Synopsis

```
    int EAIRun_Call (
       char           *lpEAIStepDefinition,
       void           *hcaseContext,
       char           *lpDelayedReleaseID,
       void          **lpAWRUniqueID)
```

Parameters

Name	Description
lpEAIStepDefinition	A NULL terminated string that contains the EAI type step definition data. This function must always check the pointer before accessing this value because if the step definition is empty, this pointer will pass as NULL.
hCaseContext	Used for the iProcess case context handle. This is required by some of the call back functions provided by the background process (such as **proEAI_GetField()**.)
lpDelayedReleaseID	A NULL terminated string containing the delayed release identifier. If the EAI step has been set up for delayed release, the external system has to store the identifier and pass it back to iProcess when it wants to release the step.
lpAWRUniqueID	Provides the return space for the asynchronous awaiting reply identifier for this external system call out. If the plug-in calls out to the external system using an 'asynchronous with reply' method, it must return a (void *) unique identifier in this parameter (i.e. *lpAWRUniqueID = (void *)my_uniqID;). That is, the value of the pointer must be unique for outstanding call-outs for the given EAI type (the pointer itself rather than the contents are used for comparison later). When the EAIRun_Getreply() function is called, it will be expected to return the AWR unique identifier returned by a previous call to this EAIRun_Call() function. This allows the background process to tell which outstanding AWR step for the EAI type a reply is received for. The main thing to note is that the call-outs for outstanding EAI steps can be returned in any order. The background process matches up the lpAWRUniqueID with the correct outstanding EAI step. Ideally, your plug-in would reply for the first call-out to complete rather than in the order of the calls.

Description

The background process calls this function in the plug-in whenever an EAI step (of the type that is handled by that plug-in) is processed in the workflow (i.e. as the action of a case start, or release / deadline expiry action of another step).

When this function is called the EAI plug-in will typically parse the step definition data supplied in **lpEAIStepDefinition** for the information that tells it:

- The ‘service’ to call on the external system that the plug-in interfaces to.
- The call-out method to be used.
- The ‘field mapping’ to use between iProcess and the external system for input / output data.
- Anything else it wants to know on a ‘per-step’ basis (such as what data to return for audit trail entries etc.).

Returns

Value	Description
EAI_ COMPLETE	Call-out successfully completed. The iProcess background process checks the delayed release setting and either processes the step or not depending on the setting.
EAI_DELAYED_ RELEASE	The call-out succeeded. The step is not to be released until the external application releases it. This overrides the EAI step generic delayed release setting defined for the step.
EAI_ASYNCH_ WITH_REPLY	The call was initiated asynchronously and the background should call EAIRun_Getreply() later to complete the call-out. This call-out method will allow the background process to process EAI steps ‘in parallel’. In most situations this will improve performance of workflows with ‘parallel paths’ that run EAI steps.
EAI_ERROR	A serious error was encountered. This will cause the background process to abort the current transaction.

Value	Description
	The EAIRun_GetLastError() function will be called to get the error description for use in any SW_WARN log or audit error entries.

EAIRun_Getreply()

Synopsis

```
int EAIRun_GetReply (
   int                *hCaseContext,
   void              **lpAWRUniqueID)
```

Parameters

Name	Description
hCaseContext	The iProcess case context handle. This value is required by some of the call back functions provided by the iProcess background process (such as proEAI_GetData()).
lpAWRUniqueID	This output parameter provides the return space for the asynchronous awaiting reply unique identifier for the external system call-out that actually replied. The background process uses this to identify the EAI step for which the reply was received.

Description

This function is optional and only needs to be defined if the plug-in supports asynchronous with reply call-out methods.

The background process calls this function when it needs to get a reply to any outstanding ‘asynch with reply’ called-out EAI steps that are still awaiting reply.

The background process only calls this function when there are ‘asynch with reply’ steps of the EAI type handled by the plug-in that have not yet replied.

The plug-in identifies the actual call-out that the reply was received for. It does this by returning the unique call-out ID that it originally returned when EAIRun_Call() was called for the step.

Note:
The following rules must be obeyed by the EAI plug-in's implementation of this function. If the plug-in cannot meet these rules then it should not use an asynch with reply call-out method:

- The implementation of EAIRun_Getreply() must block waiting for a reply (preferably the reply from the first call-out to complete).
- The blocking method used must include some form of time-out processing so that it will not remain idle if the external system does not reply.

If this function returns a value in *lpAWRUniqueID that does not match any outstanding step that the background has a record of then it is ignored. The transaction is not aborted because the background process assumes that it is a reply to a call-out that timed out during a previous transaction.

This means that you should prevent this function persistently returning invalid AWRUniqueID values because it can cause the background process to keep looping through the process.

Returns

Value	Description
EAI_COMPLETE	The call-out successfully completed. The background process will check the delayed release settings and either process the step's actions or not depending on those settings.
EAI_DELAYED_RELEASE	The call-out succeeded. Do not release the step as the external system will release it later. **Note:** This overrides the EAI Step generic delayed release settings defined for the step and forces the step to be delayed release.

Value	Description
EAI_ERROR	A serious error was encountered. This causes the background process to abort the current transaction. The EAIRun_GetLastError() function is called to get the error description for use in any SW_WARN log or audit error entries.

EAIRun_Withdraw()

Synopsis

```
int EAIRun_Withdraw (
   char               *lpEAIStepDefinition,
   char               *lpDelayedReleaseID)
```

Parameters

Name	Description
lpEAIStepDefinition	A NULL terminated string containing the EAI type specific step withdraw definition data placed in the step definition by the EAI client plug-in.
lpDelayedreleaseID	A NULL terminated string containing the delayed release unique identifier that was passed to the EAIRun_Call() function when this delayed release EAI step was sent out.

Description

The background process calls this function for outstanding delayed release EAI steps when it needs to withdraw that step (on deadline expiry, case purge etc.).

This provides the opportunity for the application definer to specify a method of removing any data held on the external system involved in the step to keep it synchronized with the iProcess case status.

You must implement this function if you want to use the CHECK_EAI _WITHDRAW_ ONPURGE process attribute. This attribute defines whether or not iProcess checks if any outstanding delayed release EAI steps have been successfully withdrawn before committing the purge transaction. iProcess uses the return value from the **EAIRun_ Withdraw()** function to determine whether it should commit (EAI_SUCCESS) or rollback (EAI_FAIL or EAI_ERROR) the purge transaction. See "CHECK_EAI_WITHDRAW_ONPURGE" in the *TIBCO iProcess Engine Administrator's Guide* for more information.

This function is not called if there is no EAI type specific step withdraw definition data defined for the step. If your plug-in and server are using Version 1 of the interface, the server will call this function.

Refer to EAIRun_Withdraw_v2() for information about the new withdraw function available in the Version 2 interface
specification.

Note: Asynchronous with reply call-outs are not supported. This means that when this function returns, the background process considers the withdrawal to be complete.

Returns

Value	Description
EAI_ SUCCESS	Success.
EAI_ERROR	Failed. The background process aborts the transaction. The EAIRun_ GetLastError() function is called to get the error description used in any SW_WARN log or audit error entries.

EAIRun_Withdraw_v2()

Synopsis

```
int EAIRun_Withdraw_v2 (
   char               *lpEAIStepDefinition,
   char               *lpDelayedReleaseID,
   void               *hCaseContext)
```

Parameters

Name	Description
lpEAIStepDefinition	A NULL terminated string containing the EAI type specific step withdraw definition data placed in the step definition by the EAI client plug-in.
lpDelayedreleaseID	A NULL terminated string containing the delayed release unique identifier that was passed to the EAIRun_Call() function when this delayed release EAI step was sent out.
hCaseContext	The iProcess case context handle. This value is required by some of the call back functions provided by the iProcess background process (such as proEAI_GetData()).

Description

If your server supports Version 2 (or higher) of the EAI run-time interface and the plug-in itself supports it, this function is used rather than the EAIRun_Withdraw() function used in Version 1 of the interface. The difference between this function and EAIRun_Withdraw() is that this function is passed a case context parameter (*hCaseContext*) so it can utilize the background call back functions such as proEAI_GetData() and proEAI_SetData(). This means the function has access to the case field values.

The background process calls this function for outstanding delayed release EAI steps when it needs to withdraw that step (on deadline expiry, case purge etc.)

This provides the opportunity for the application definer to specify a method of removing any data held on the external system involved in the step to keep it synchronized with the iProcess case status.

You must implement this function if you want to use the CHECK_EAI _WITHDRAW_ ONPURGE process attribute. This attribute defines whether or not iProcess checks if any outstanding delayed release EAI steps have been successfully withdrawn before committing the purge transaction. iProcess uses the return value from the **EAIRun_ Withdraw()** function to determine whether it should commit (EAI_SUCCESS) or rollback (EAI_FAIL or EAI_ERROR) the purge transaction. See “CHECK_EAI_WITHDRAW_ONPURGE” in the *TIBCO iProcess Engine Administrator’s Guide* for more information.

This function is not called if there is no EAI type specific step withdraw definition data defined for the step.

Note: Asynchronous with reply call-outs are not supported. This means that when this function returns, the background process considers the withdrawal to be complete.

Returns

Value	Description
EAI_ SUCCESS	Success.
EAI_ERROR	Failed. The background process aborts the transaction. The EAIRun_ GetLastError() function is called to get the error description used in any SW_WARN log or audit error entries.

EAIRun_GetLastError()

Synopsis

```
int EAIRun_GetLastError (
   char                  *lpErrorText,
    long                    *cbErrorText)
```

Parameters

Name	Description
lpErrorText	Text description of the last error that occurred when the background process called any **EAIRun_**() functions in the plug-in.
cbErrorText	Maximum number of bytes that this function is allowed to write to the lpErrorText parameter, including the NULL terminator.

Description

The background calls this function after a call to one of the other **EAIRun_** () functions has returned an error.

The background process typically uses the error text as part of an SW_WARN message it logs because of the failure.

Returns

Value	Description
Integer	Plug-in specific error code.

EAIRun_CallCleanUp()

Synopsis

```
int EAIRun_CallCleanUp (
   int                  Aborting)
```

Parameters

Name	Description
Aborting	This parameter is non-zero if the background process calling this function knows that there has been an error during the workflow processing within the current transaction.

Description

The background process calls this function at the end of processing a workflow as far as it can be for a given case in the current transaction.

This gives the plug-in the opportunity to ensure that everything has been cleaned up properly. The plug-in can also indicate to the background process that something has gone wrong and that the transaction should be aborted. For example, if there are asynch with reply call-outs that have not received a reply.

Returns

Value	Description
EAI_SUCCESS	Success.
EAI_ERROR	Error. The transaction is aborted.

EAIRun_GetReleaseVers()

Synopsis

```
void EAIRun_GetReleaseVers (
   char                     *lpRelVers,
   int                       cbmax)
```

Parameters

Name	Description
lpRelVers	Return data buffer for release version string.
cbmax	Maximum number of bytes that can be written to **lpRelVers**, including the NULL terminator.

Description

This function returns the release version string for the plug-in in the **lpRelVers** parameter.

Returns

None.

proEAI_GetCallBack()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
void *proEAI_GetCallBack (
   int                    fnIDX)
```

Parameters

Name	Description
fnIDX	A call back function identifier, which determines the call back function pointer to return.

Description

This function returns a pointer to various call-back functions that are required by the EAI plug-in.

The address of this function is passed to the EAIRun_Init() function when the background process initializes the plug-in. The following table details the call-back functions that can be accessed using this function, the value of the **fnIDX** parameter to use, and the type definition that this function returns.

Call Back Function	fnIDX	Return Type
proEAI_Debug()	PROEAI_ DEBUG	PROEAI_DEBUG_FN
proEAI_GetData()	PROEAI_ GETDATA	PROEAI_GETDATA_FN
proEAI_SetData()	PROEAI_ SETDATA	PROEAI_SETDATA_FN
proEAI_EvaluateExpr()	PROEAI_ EVALEXPR	PROEAI_EVALEXPR_FN
proEAI_DescribeData()	PROEAI_ DESCRIBEDATA	PROEAI_DESCRIBEDATA_FN
proEAI_FreeData()	PROEAI_ FREEDATA	PROEAI_FREEDATA_FN

In the EAIRun_Init() function, the plug-in can either store this function address (in some persistent memory to use later to access the other call-back functions) or use it to get the addresses of all the call-back functions it may need later (and store those in persistent memory).

Returns

Value	Description
NOT NULL	The address of the call-back function identified by fnIDX.
NULL	Invalid fnIDX value.

proEAI_Debug()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
void *proEAI_Debug (
   int              Level,
   char            *lpFormat, ...)
```

Parameters

Name	Description
Level	Combination of bits (as defined by **EAIDBG_** definitions in **eairun.h**) declaring which debug levels the given message should be output at.
lpFormat	This is a **printf ()** equivalent format string (i.e. “String: %s, int: %d”), each ‘%’ variable insertion sequence must have a corresponding argument of the correct type in the variable argument list).
...	Variable argument parameters (use like **printf ()** variable arguments).

Description

This function builds a message from the given format string and variable parameters (in the same way as a **printf ()** call) to the iProcess background debug log.

The message is only output to the debug log if the background process debug setting (in the **process_attribute** table) indicates that the given Level(s) are required.

EAI plug-in implementations should use the proEAI_Debug() call-back function to log trace/debug messages as follows:

- EAIDBG_PROC – Each **EAIRun_ ()** function implemented by the plug-in should log a message at this level on entry / exit.
- The exit message of each function should indicate success or failure.
- All error conditions should log a message at EAIDBG_ERR level.
- All messages should contain the **EAIRun_ ()** function name involved and the EAI type name. This is because all EAI plug-in's output messages under debug area 'x' and the administrator needs to distinguish messages between different plug-ins.
- For other standard EAIDBG_ levels see the `eairun.h` file.

This call-back function is accessed via the proEAI_GetCallBack() function passed to the plug-in's EAIRun_Init() function:

```
    PROEAI_DEBUG_FN       proEAI_Debug;
    proEAI_Debug = (PROEAI_DEBUG_FN) proEAI_GetCallBack    (PROEAI_DEBUG);
```

The **proEAI_Debug** variable can then be used like any other C function:

```
    proEAI_Debug (EAIDBG_PROC, “EAIRun_Call (): Entering
    function”);
    NumCalls++;
    proEAI_Debug (EAIDBG_SUBPROC1, “EAIRun_Call (): Number of    call-outs:
  %d”, NumCalls);
```

Returns

None.

proEAI_DescribeData()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int proEAI_DescribeData (
   void                  *hCaseContext,
   char                  *lpFldName,
   char                  *lpFldType,
   int                   *lpFldLen,
   int                   *lpFldDpMid)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in that it is legal to use this call back from.
lpFldName	Input NULL terminated string name of field required.
lpFldType	Output of single character field type. Possible field types are specified in `eairun.h` as the **EAI_FTYPE_** definitions. If not required, this parameter can be passed as NULL.
lpFldLen	Length of field in bytes not including the NULL terminator: • This is the defined maximum length of the field and not the length of its current value. • This is not used for MEMO fields. • For numeric fields with decimal places this includes the decimal places and the decimal point. For example, a field whose format is 9999.99 will have a length of 7 with 2

Name	Description
	decimal places. If this is not required, the parameter can be passed as NULL.
lpFldDpMid	For numeric fields: the number of decimal places. For Memo fields: the memo field identifier. If this is not required, this parameter can be passed as NULL.

Description

This function allows the EAI plug-in to access the definition of the given iProcess case data field.

The **hCaseContext** parameter should be the same as the value passed to the EAI plug-in's EAIRun_Call() or EAIRun_Getreply() functions as a parameter. The **hCaseContext** must not be stored and used outside of the functions to which it is passed as a parameter because you will have undefined results.

This call-back function is accessed via the proEAI_GetCallBack() function passed to the plug-in's EAIRun_Init() function:

```
    PROEAI_DESCRIBEDATA_FN                proEAI_DescribeData;
    proEAI_DescribeData = (PROEAI_DESCRIBEDATA_FN)    proEAI_GetCallBack (
       PROEAI_DESCRIBEDATA);
```

The **proEAI_DescribeData** variable can then be used like any other C function:

```
    char      RetCode;
    char      FldType;
    int       FldLen;
    int       FldDpMid;
    RetCode = proEAI_DescribeData (hCaseContext,
                                   “SW_CASEDESC”,
                                   &FldType,
                                   &FldLen,
                                   &FldDpMid);
    … do stuff with field description i.e. validate type against
      external system field that it is mapped to  …
```

Note: iProcess attachment fields are not supported in this version of the EAI run-time plug-in interface. If the field requested is an attachment, this function returns an error.

Returns

Value	Description
EAI_SUCCESS	Success.
EAI_ERROR	Error. The field does not exist or the case context handle is invalid.

proEAI_GetData()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int proEAI_GetData (
   void             *hCaseContext,
   char             *lpFldName,
   char             *lpFldType,
   int              *lpFldLen,
   int              *lpFldDpMid,
   long             *lpFldValueLen)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in

Name	Description
	that it is legal to use this call back from.
lpFldName	Input NULL terminated string name of field required.
lpFldType	Output of single character field type. Possible field types are specified in `eairun.h` as the **EAI_FTYPE_** definitions. If not required, this parameter can be passed as NULL.
lpFldLen	Length of field in bytes not including the NULL terminator: • This is the defined maximum length of the field and not the length of its current value. • This is not used for MEMO fields. • For numeric fields with decimal places this includes the decimal places and the decimal point. For example, a field whose format is 9999.99 will have a length of 7 with 2 decimal places. If not required, this parameter can be passed as NULL.
lpFldDpMid	For numeric fields: the number of decimal places. For Memo fields: the memo field identifier. If not required, this parameter can be passed as NULL.
lpFldValueLen	Length of the dynamically allocated data buffer returned by this function, including the NULL terminator. If not required, this parameter can be NULL.

Description

This function enables the EAI plug-in to access the definition and current value of the given iProcess case data field.

The **hCaseContext** parameter should be the same as the value passed to the EAI plug-in's EAIRun_Call() or EAIRun_Getreply() functions as a parameter. The **hCaseContext** must not be stored and used outside of these two functions, doing so will have undefined results.

Only use the **proEAI_GetData ()** function from within an **EAIRun_ ()** function that is itself passed as a **hCaseContext** parameter.

On success, the address of the current field value is returned from this function. This is a NULL terminated string (all iProcess case fields are presented as strings regardless of their type). This function dynamically allocates the memory for field value data returned by this function.

Note: It is the responsibility of the caller of this function to free this dynamically allocated buffer (using the proEAI_FreeData() function) when it has finished with the field value. If this is not done, it will result in memory leaks in the background process.

This call-back function is accessed via the proEAI_GetCallBack() function passed to the plug-in's EAIRun_Init() function:

```
   PROEAI_GETDATA_FN   proEAI_GetData;
   proEAI_GetData = (PROEAI_GETDATA_FN) proEAI_GetCallBack    (PROEAI_
GETDATA);
```

The **proEAI_GetData** variable can then be used like any other C function:

```
   char      *lpFldValue;
   char       FldType;
   int        FldLen;
   int        FldDpMid;
   long       FldValueLen;
   lpFldValue = proEAI_GetData (hCaseContext,
                                “SW_CASEDESC”,
                                &FldType,
                                &FldLen,
                                &FldDpMid,
                                &FldValueLen);
   … do stuff with field value i.e. pass on to external system …
   free (lpFldValue);
```

Note: iProcess attachment fields are not supported in this version of the EAI run-time plug-in interface. If the field requested is an attachment then this function returns an error.

Returns

Value	Description
Not NULL	Value of the requested field. The other return parameters are assigned their respective values (if not passed as NULL). The caller should free this dynamically allocated memory by passing the returned pointer to the proEAI_FreeData() call-back function.
NULL	Error. The field does not exist or the case context handle is invalid.

proEAI_FreeData()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int proEAI_FreeData (
   void              *hCaseContext,
   char               FldType,
   char              *lpFldValueBuffer)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in that it is legal to use this call back from.
FldType	Type of field that data is being freed for.
lpFldValueBuffer	Input NULL terminated string name of field required.

Description

This function should be called to free each field value data buffer allocated by the proEAI_GetData().

The address of this function can be accessed via the proEAI_GetCallBack() function address passed to the EAIRun_Init() function in the plug-in.

Returns

Value	Description
EAI_SUCCESS	Success.
EAI_ERROR	Invalid field data buffer.

proEAI_SetData()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int proEAI_SetData (
   void            *hCaseContext,
   char            *lpFldName,
   char            *lpFldValue,
   long             FldValueLen)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in that it is legal to use this call back from.
lpFldName	Input NULL terminated string name of field required.
lpFldValue	Input value to set case data field to. For all but attachment fields this should be a pointer to a NULL terminated string. For memo fields this can be up to 2 GB of data. For other field types this must not be longer than the field length. If the value of this parameter is NULL (i.e. an empty string), or the **FldValueLen** is 0 then the case data field is set to SW_NA. **Note:** All field values in iProcess are represented as alphanumeric strings regardless of their type (except attachments).
FldValueLen	Length of the data in the **lpFldValue** buffer (including any NULL terminator).

Description

This function allows the EAI plug-in to set the given iProcess case data field to the given value.

The **hCaseContext** parameter should be the same as the value passed to the EAI plug-in’s EAIRun_Call() or EAIRun_Getreply() functions as a parameter. The **hCaseContext** must not be stored and used outside of these two functions otherwise you will have undefined results. Only use the proEAI_SetData() function from within an **EAIRun_** () function that is itself passed as a **hCaseContext** parameter.

This call-back function is accessed via the proEAI_GetCallBack() function passed to the plug-in’s EAIRun_Init() function:

```
   PROEAI_SETDATA_FN     proEAI_SetData;
   proEAI_SetData = (PROEAI_SETDATA_FN) proEAI_GetCallBack    (PROEAI_
SETDATA);
```

The **proEAI_SetData** variable can then be used like any other C function:

```
   char      myData[100];
   /* Next line would typically be ‘set data from ext system’ */
   strcpy (myData, “Hello World”);
   lpFldValue = proEAI_SetData (hCaseContext,
                               “TEXTFLD1”,
                                myData,
                                strlen (myData)+1);
```

Note: iProcess attachment fields are not supported in this version of the EAI run-time plug-in interface. If the field requested is an attachment, this function returns an error.

Returns

Value	Description
EAI_SUCCESS	Success.
EAI_ERROR	Invalid field name or value.

proEAI_EvaluateExpr()

Note: This function is not defined by the plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int proEAI_EvaluateExpr (
   void                  *hCaseContext,
   char                  *lpExpression,
   void                  *lpReturnValue,
   long                   cbMaxReturnValue)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in that it is legal to use this call back from.
lpExpression	Input iProcess expression.
lpReturnValue	The return value of the expression, the data assigned to this data buffer depends on the return code - see the Returns section for the raw data types for given return codes. It is recommended that the return buffer is at least 256 bytes long.
cbMaxReturnValue	Length of the data that can be assigned to the **lpReturnValue** buffer.

Description

This function evaluates the given iProcess expression (the same kind of expression that can be used in iProcess forms, conditions and scripts) and returns the value of that expression.

Note: For BOOLEAN (condition) expressions, the expression text should not contain the “if” statement (i.e. use “FLD1 = 1” rather than “if (FLD1 = 1)”.

This function enables EAI plug-ins to use their own expressions involving iProcess data and saves the necessity of them creating their own expression evaluator software.

The only expression types supported are those that evaluate to BOOLEAN, TEXT, NUMERIC, DATE or TIME. Other types will return an error. Note that BOOLEAN expression types assign no data to the return value buffer.

Returns

Value	Description
EAI_EXPR_ BOOL_FALSE	The expression is boolean and evaluated to FALSE. **lpReturnValue** is not assigned.
EAI_EXPR_ BOOL_TRUE	The expression is boolean and evaluated to TRUE. **lpReturnValue** is assigned.
EAI_EXPR_ TEXT	The expression evaluated to a text value. **lpReturnValue** is a NULL terminated string.
EAI_EXPR_ NUMERIC	The expression evaluated to a numeric value. **lpReturnValue** is a NULL terminated string representing a numeric (with 6 decimal places).
EAI_EXPR_ DATE	The expression evaluated to a date value. **lpReturnValue** is a NULL terminated string in the standard date format defined for the iProcess installation.
EAI_EXPR_TIME	The expression evaluated to a time value. **lpReturnValue** is a NULL terminated string in the standard time format defined for the iProcess installation.
EAI_EXPR_ SWNA	The expression evaluates to SW_NA (not assigned). **lpReturnValue** is not assigned.
EAI_EXPR_VAL_ TOOBIG	The value that the expression evaluates to is too large to fit into the return **lpReturnValue** buffer.

Value	Description
	lpReturnValue is not assigned.
EAI_EXPR_ INVALID	Invalid expression, or expression returns a type not supported by this interface or other serious error. **lpReturnValue** is not assigned.
EAI_ERROR	Invalid parameter or other system error.

proEAI_GetDateLine()

Note: This function is not defined by the EAI plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
void proEAI_GetDateLine (
   int                     type,
   char                   *lpBuf,
   int                     bufsize,
```

Parameters

Name	Description
type	Must be set to zero to return the default date format field. Other values are reserved for future use and are unsupported.
lpBuf	A pointer to a buffer to hold the return data.
bufsize	The size of the buffer allocated by the caller.

Description

This function returns the date format string from the DATELINE entry of the `SWDIR\etc\staffpms` file, which is cached when the server EAI plug-in initializes.

Returns

Typically it returns a null terminated string such as "dmy". However, the exact value depends on the locale of the installed iProcess server.

Any failure in the function results in a null string being returned in the buffer.

proEAI_GetDataEx()

Note: This function is not defined by the EAI plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int proEAI_GetDataEx (
   void          *hCaseContext,
   char          *lpFldName,
   char          *lpFldType,
   int           *lpFldLen,
   int           *lpFldDpMid,
   long          *lpFldValueLen,
   int           *lpFldFlags)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in that it is legal to use this call back from.

Name	Description
lpFldName	Input NULL terminated string name of field required.
lpFldType	Output of single character field type. Possible field types are specified in `eairun.h` as the **EAI_FTYPE_** definitions. If not required, this parameter can be passed as NULL.
lpFldLen	Length of field in bytes not including the NULL terminator: • This is the defined maximum length of the field and not the length of its current value. • This is not used for MEMO fields. • For numeric fields with decimal places this includes the decimal places and the decimal point. For example, a field whose format is 9999.99 will have a length of 7 with 2 decimal places. If not required, this parameter can be passed as NULL.
lpFldDpMid	For numeric fields: the number of decimal places. For Memo fields: the memo field identifier. If not required, this parameter can be passed as NULL.
lpFldValueLen	Length of the dynamically allocated data buffer returned by this function, including the NULL terminator. If not required, this parameter can be NULL.
lpFldFlags	A pointer to the memory space for the return of a value that determines whether the field is an array field. If the EAI_FFLAG_ARRAY (value 0x0010) bit in the returned value is set, the field is an array field. If it is not set, the field is not an array field.

Description

This function enables the EAI plug-in to access the definition and current value of the given iProcess case data field.

The **hCaseContext** parameter should be the same as the value passed to the EAI plug-in's EAIRun_Call() or EAIRun_Getreply() functions as a parameter. The **hCaseContext** must not be stored and used outside of these two functions, doing so will have undefined results. Only use the **proEAI_GetDataEx ()** function from within an **EAIRun_ ()** function that is itself passed as a **hCaseContext** parameter.

On success, the address of the current field value is returned from this function. This is a NULL terminated string (all iProcess case fields are presented as strings regardless of their type). This function dynamically allocates the memory for field value data returned by this function.

Note: It is the responsibility of the caller of this function to free this dynamically allocated buffer (using the proEAI_FreeData() function) when it has finished with the field value. If this is not done, it will result in memory leaks in the background process.

This call-back function is accessed via the proEAI_GetCallBack() function passed to the plug-in's EAIRun_Init() function:

```
   PROEAI_GETDATAEX_FN   proEAI_GetData;
   proEAI_GetData = (PROEAI_GETDATAEX_FN) proEAI_GetCallBack    (PROEAI_
GETDATAEX);
```

The **proEAI_GetDataEx** variable can then be used like any other C function:

```
   char      *lpFldValue;
   char       FldType;
   int        FldLen;
   int        FldDpMid;
   long       FldValueLen;
   lpFldValue = proEAI_GetDataEx (hCaseContext,
                                  "SW_CASEDESC",
                                  &FldType,
                                  &FldLen,
                                  &FldDpMid,
                                  &FldValueLen);
   … do stuff with field value i.e. pass on to external system …
   free (lpFldValue);
```

Note: iProcess attachment fields are not supported in this version of the EAI run-time plug-in interface. If the field requested is an attachment then this function returns an error.

Returns

Value	Description
Not NULL	Value of the requested field. The other return parameters are assigned their respective values (if not passed as NULL). The caller should free this dynamically allocated memory by passing the returned pointer to the proEAI_FreeData() call-back function.
NULL	Error. The field does not exist or the case context handle is invalid.

proEAI_DescDataEx()

Note: This function is not defined by the EAI plug-in. It is provided by the iProcess background process code at run-time and can be called by the plug-in.

Synopsis

```
int ProEAI_DescDataEx (
      void          *hCaseContext,
      char          *lpFldName,
      char          *lpFldType,
      int           *lpFldLen,
      int           *lpFldDpMid,
      int           *lpFldFlags)
```

Parameters

Name	Description
hCaseContext	The case context handle. This is passed to functions in the plug-in that it is legal to use this call back from.
lpFldName	Input NULL terminated string name of field required.
lpFldType	Output of single character field type. Possible field types are specified in `eairun.h` as the **EAI_FTYPE_** definitions. If not required, this parameter can be passed as NULL.
lpFldLen	Length of field in bytes not including the NULL terminator: • This is the defined maximum length of the field and not the length of its current value. • This is not used for MEMO fields. • For numeric fields with decimal places this includes the decimal places and the decimal point. For example, a field whose format is 9999.99 will have a length of 7 with 2 decimal places. If this is not required, the parameter can be passed as NULL.
lpFldDpMid	For numeric fields: the number of decimal places. For Memo fields: the memo field identifier. If this is not required, this parameter can be passed as NULL.
lpFldFlags	A pointer to the memory space for the return of a value that determines whether the field is an array field. If the EAI_FFLAG_ARRAY (value 0x0010) bit in the returned value is set, the field is an array field. If it is not set, the field is not an array field.

Description

This function allows the EAI plug-in to access the definition of the given iProcess case data field.

The **hCaseContext** parameter should be the same as the value passed to the EAI plug-in's EAIRun_Call() or EAIRun_Getreply() functions as a parameter. The **hCaseContext** must not be stored and used outside of the functions to which it is passed as a parameter because you will have undefined results.

This call-back function is accessed via the proEAI_GetCallBack() function passed to the plug-in's EAIRun_Init() function:

```
PROEAI_DESCDATAEX_FN          proEAI_DescDataEx;
proEAI_DescDataEx = (PROEAI_DESCDATAEX_FN)    proEAI_GetCallBack (
   PROEAI_DESCDATAEX);
```

The **proEAI_DescDataEx** variable can then be used like any other C function:

```
char      RetCode;
char      FldType;
int       FldLen;
int       FldDpMid;
RetCode = proEAI_DescDataEx (hCaseContext,
                             "SW_CASEDESC",
                              &FldType,
                              &FldLen,
                              &FldDpMid);
… do stuff with field description i.e. validate type against
  external system field that it is mapped to  …
```

Note: iProcess attachment fields are not supported in this version of the EAI run-time plug-in interface. If the field requested is an attachment, this function returns an error.

Returns

Value	Description
EAI_SUCCESS	Success.

Value	Description
EAI_ERROR	Error. The field does not exist or the case context handle is invalid.

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® Plug-in SDK 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® Plug-in SDK, 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