Argus Interware Inc.

# Argus ONE Plug-In Extensions (PIEs )Technical Notes

## Overview

Argus-PIE is an empowering technology which enables users and Developers/VARs of the Argus Open Numerical Environments (Argus ONE) to tailor and extend the Argus technology for their special needs. The Argus-PIE technology is based on Argus Open Numerical Environments' inherent capabilities and allows developers to tap into Argus' powerful tools from external code. It was developed by Argus to allow developers to easily and quickly offer and market their advanced technologies in numerical modeling, geostatistics, 3D visualization and other areas, to the rapidly growing community of Argus Open Numerical Environments users as well as to their own customers.

Developing and using Argus-PIEs is accomplished without modifying the Argus Open Numerical Environments, which is an "Off-the-shelf" application, but rather by programming small C, C++, Fortran or JAVA functions, linking them as DLLs/Shared-Libraries, and dropping them into a special directory from which they are seamlessly linked to Argus' products.

These Plug-ins can be distributed and immediately used by any user of the Argus Open Numerical Environments.

Note: Argus-PIEs should be used when Argus ONE built-in capabilities like expressions and export templates, are not sufficient.

### Types of Argus-PIEs

Argus-PIEs are categorized according to the functionality they allow you add to Argus ONE and to the Argus ONE internal data they allow you to access. Argus is continuously opening its products and publishing additional Application Programming Interfaces (APIs). Currently four Argus-PIE types are available. A single Argus-PIE may include as many PIEs of each of the available PIE types, thus allowing one to develop an application that fully integrates Argus ONE and his/her product.

#### Function PIEs

A function takes arguments (integers, reals, strings, booleans) and returns a value. A function is accessible from any Argus Expression, including Layer parameters and export templates expressions.

- PIE functions install menu items for themselves in the expression function list.
- PIE functions can be used for units conversions, coordinate system transformations, etc.
- PIE functions are invoked when an Argus expression they appear in is evaluated.

#### Export Template PIEs

An Export Template PIE can define and store, (hidden if necessary), model specific export templates.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

- An Export Template PIE installs a sub-menu item under the Export menu of an Argus ONE layer, and when invoked, causes the Argus ONE layer it is invoked from to export the data specified in the template, and perform any other operations imbedded within it, such as running an external code.
- Export Template PIEs can be used to export data from Argus ONE layers in different formats, to launch external codes, to create a library of export templates for various models, etc.
- Export Template PIEs are invoked when the user pulls the Export sub-menu item they install.

### Project PIEs

A Project PIE allows one to create a model specific Argus ONE project having a custom layer structure which is automatically created. The Project PIE enables developers to launch dialogs to allow the user to input model specific control parameters (such as Non Spatial-dependent variables). The Project PIE can also save its internal data structure within the Argus Project it creates, and load it when such a project reloads.

- Project PIEs install a New Project sub-menu item in the File menu and an Edit Project Info sub-menu item in the Edit menu.
- Project PIEs can be used to fully integrate numerical models within the Argus Environments and to automate data/layer structures creation.
- Project PIEs are invoked from the File/New sub-menu item and in the Edit menu to allow the user to invoke the Control Parameters dialogs. Any number of Project PIEs can be installed as FILE/NEW sub-menus.

### Import PIEs

An Import PIE can import data to any layer it is invoked from. Data can be read from one of the following sources: any file, another layer, or an internal algorithm.

- Import PIEs install sub-menus in the Import menu item under the File menu.
- Import PIEs can be used to translate data in formats other then Argus generic formats, to perform coordinate transformations while importing such data, to operate geostatistical algorithms while importing information, etc.
- Import PIEs are invoked when the user selects the PIE's Import sub-menu.

### Interpolation PIEs

An Interpolation PIE allows one to customize the way Argus ONE interpolates data in the Data or Information (contour) layers.

- Interpolation PIEs install a new items in the Layers' Dialog, allowing the user to select one.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

- The selected Interpolation PIE receives all the data points (or contour vertices) in the layer, in order to prepare it's internal data tables.
- When a calculation of interpolated value is required, the Interpolation PIE is called again with the appropriate *x*, and *y* values, returning the interpolated value.

## Argus ONE Call-Back Functions

In order to allow the PIE programmers to access Argus' internal capabilities, a set of call-back functions is supplied for the PIE programmers, which can assist them in their work. These functions covers areas such as:

- Creating and removing layers and parameters.
- Importing and exporting objects from or to various layers.
- Allocating memory.
- Handling of files and directories.
- Reading and setting layers', parameters' and objects' properties.
- Direct manipulation of various objects.
- Interacting with the user.
- Displaying progress-bar for long operations.
- And more...

The call-back funcions are listed in the header file `ANECB.H`. Please consult this file for an up-to-date list of available functions and their usage details.

## Multi-Platform and Platform-Specific solutions

Any one of the Argus-PIE types defined above may be written as a multi-platform solution, that is, a single c/c++ source code that can be compiled on several target platforms. Since Argus ONE is by itself a multi-platform application it is beneficial to write multi-platform PIEs.

In some cases, the use of an operating system's special features may be required. Such features may include GUI specific features (MFC, OWL, MacApp, Motif, and any other GUI tool kit or framework), 3D graphics, and access to peripheral devices (data acquisition).

Argus Interware is developing a basic GUI as part of the Argus Environments to allow developers to develop multi-platform Argus-PIEs, thus reducing the number of instances where platform specific code needs to be written.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

## PIE-Libraries - Integrating Several Argus-PIEs

Argus-PIE libraries may include some, or all of the PIE types described above. For example, a single PIE library may integrate a Project-PIE which will create a new document for your model, an Export Template PIE which will export data to your model and run it, an Import PIE which will read your model output back into Argus ONE, and PIE-Functions for accessing the various fields of the Project-PIE data structure, or to support some special conversions.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

Argus Interware Inc.

# Argus Open Numerical Environments Plug-In Extension Technology

## Technical Notes (V 2.0 - May, 1999)

### Overview

This document describes in detail how to develop Argus-PIEs and how Argus-PIEs communicate with Argus Open Numerical Environments (Argus ONE). In this document you will find formal definitions, as well as examples. You can use these examples as templates for creating your own PIEs.

When Argus ONE is launched it searches for Argus-PIE libraries, loads them into memory and initiates the PIEs. The initialization is performed when Argus ONE calls a special function within the PIE-library which is named `GetANEFunctions`. This function should return all the information required by Argus ONE so that it can use the various PIEs contained in the library. This information is comprised of a descriptor record for each PIE.

The descriptor record includes, among other things, Vendor and Product names. These are very important fields, as they will help avoid possible collisions between different products your company develops as well as products developed by other companies.

Also included in the descriptor record is its version number. As Argus continues to develop its products and PIE technology, new additions will be introduced to the descriptors. At each such instance Argus might increment the record version number. When Argus ONE gets the descriptor record it uses the version field to properly handle PIEs developed using older versions.

#### Naming Conventions

Below are several conventions we have used throughout the Argus ONE interface and examples:

- Enumeration type names are prefixed by capital `E` as in `EPIEType`
- Constants are prefixed by lower-case `k` as in `kFunctionPIE`
- Basic types are declared in capital letters as `ANE_INT32`

### Basic data types definitions

Argus has defined several data types for interfacing with Argus ONE. Use these data types when writing PIEs. These data types allow you to use the same variable declarations across different platforms.

#### ANE typedefs:

- `ANE_INT32` 32 bit integer



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

- `ANE_INT16` 16 bit integer
- `ANE_BOOL` boolean (implemented as int)
- `ANE_DOUBLE` double precision floating point number
- `ANE_STR` null terminated string (actually a `char*`)
- `ANE_CSTR` constant null terminated string (actually a `const char*`)
- `ANE_PTR` a generic pointer - implemented as `void*`
- `ANE_CPTR` a constant pointer - implemented as `cosnt void*`

## Argus PIE Descriptor and related types

The `ANEPIEDesc` is the basic descriptor record needed to describe each and any Argus-PIE. It contains the PIE name, vendor and product names as well as the PIE kind and a pointer to a record of additional information. The record of additional information is PIE-type specific. The various types of additional information records are discussed later in this document.

Below are several definitions of enumerations that are defined for your convenience. Use these enumerated types in the various fields as necessary:

### `EPIEType` Enumeration

The `EPIEType` enumerates the different types of PIEs, as more types and APIs are defined by Argus, the `EPIEType` enumeration will be extended to support these additional types:

- `kFunctionPIE` Function Plug In Extension
- `kExportTemplatePIE` Export Template Plug In Extension
- `kProjectPIE` Project Plug In Extension
- `kImportPIE` Import Plug In Extension
- `kInterpolationPIE` Interpolation Plug In Extension
- `kRenumberPIE` An extension for node-renumbering algorithm (not implemented yet)

### `EPIELayerType` Enumeration

The `EPIELayerType` enumerates the different types of Argus layers:

- `kPIETriMeshLayer` Triangular Finite Elements Mesh Layer
- `kPIEQuadMeshLayer` Quadrilateral Finite Elements Mesh Layer
- `kPIEInformationLayer` Information Layer
- `kPIEGridLayer` Finite Difference Grid Layer
- `kPIEDataLayer` Static Data Layer
- `kPIEMapsLayer` Maps (Grapihcs) Layer
- `kPIEDomainLayer` Domain Outline Layer
- `kPIEGroupLayer` Group Layer
- `kPIEAnyLayer` Denotes compliance with any type of layer



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

### `ENumberType` Enumeration

The `ENumberType` enumerates the different value types PIE-Functions can handle. Use it to inform Argus ONE the type of the parameters your PIE-Function expects to receive, as well as the type of value your PIE-function will return:

- `kPIEBoolean` Boolean type, transferred as ANE_BOOL
- `kPIEInteger` Integer type, transferred as ANE_INT32
- `kPIEFloat` Float type, transferred as ANE_DOUBLE
- `kPIEString` String type, transferred as ANE_STR

#### Constants

`ANE_PIE_VERSION` The version of the Argus ONE PIE interface. If you use this constant to initialize the `version` field of `ANEPIEDesc`, recompiling the source with future addition of the PIE interface will result PIE that is compatible with that latest version.

### `ANEPIEDesc` Structure

The `ANEPIEDesc` structure which is the main descriptor of a PIE, includes the following fields:

`version` Type: `ANE_INT32`
The version of the `ANEPIEDesc` structure. This field is needed for Argus to be able to improve the API in the future and to add more fields to the descriptor. If you use the constant `ANE_PIE_VERSION` to initialize this field, you can be sure that the version is correct when you recompile your PIE using a future addition of the PIE interface.

`vendor` Type: `ANE_STR`
The name of the vendor. In the future, Argus will use this field to allow Argus ONE to detect name collisions of PIEs from different vendors (Still not implemented).

`product` Type: `ANE_STR`
A unique name of the product (for a specific vendor). You should fill it with your product name, so multiple products can be supported. (Multiple product tests still not implemented)

`name` Type: `ANE_STR`
The name of the PIE. This field has different meanings for different types of PIEs, but in general it should be unique for each type of PIE that you define, that is, you can have an Import PIE and export PIE with the same name, but



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

| | |
|---|---|
| | not two Import PIEs with the same name. (See PIE specific explanations below) |
| `type` | Type: `EPIEType`<br>The type of the PIE. See description of `EPIEType` above for available types of PIEs. |
| `descriptor` | Type: `ANE_PTR`<br>A pointer to the PIE-specific additional information record. Each PIE type has a specific descriptor, which contains additional information about it. Those descriptors are discussed in detail below. |

**Extension possibility**

New types of PIEs can be added in the future without effecting existing PIEs.

If for some reason, new fields are added to `ANEPIEDesc`, the `version` field ensures that old PIEs won't have to change, unless the developer chooses to use the new features.

### Creating a PIE Library

To Add a PIE to Argus ONE, follow the steps below:

- Write a program module (using your programming language of choice) with various functions that implement the different task of the PIEs.
- Create (as a static, or in the heap) `ANEPIEDesc` structure for each of the of PIEs.
- Create (as a static, or in the heap) additional information records for each PIE you want to implement (see specific sections for `FunctionPIEDesc`, `ExportTemplatePIEDesc`, `ImportPIEDesc` and `ProjectPIEDesc` below)
- Define the interface function `GetANEFunctions`. This function is called by Argus ONE when the PIE is loaded and it should return an array of pointers to the different `ANEPIEDesc`s that you have created:
  - ◊ In "C" use `void GetANEFunctions(ANE_INT32* numNames, ANEPIEDesc*** descriptors)`
  - ◊ In "C++" use `void GetANEFunctions(ANE_INT32& numNames, ANEPIEDesc**& descriptors)`
  - ◊ `numNames` should be set to the number of PIEs returned
  - ◊ `descriptors` should be set to an array of `ANEPIEDesc*`s (`ANEPIEDesc` pointers) that points to the different `ANEPIEDesc` records for each PIE
- Create an array of `ANEPIEDesc*` with enough elements for the PIEs you want to add
- Fill the array with pointers to the descriptors of the PIEs you have created
- Set the numName and descriptors parameters with the number of the PIEs and the array you created



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

◊ In “C++”:
`numNames =` *number of name*
`descriptors =` *the array*
◊ in “C”:
`*numNames =` *number of name*
`*descriptors =` *the array*

- Make sure that the function `GetANEFunctions` is available exported as an external symbol (see platform specific section)
- Compile the program and create a library:
  on Windows 95 and Windows NT create a DLL
  on Power Macintosh create a Shared Library
  on Unix Workstation create a Shared Object
- Copy the created library to the correct directory
  on Windows, ANEPLUGS
  On Unix Workstations, ANEPlugIns
  On PowerMac any folder below the folder the Argus ONE application resides in

## Function PIE Descriptor and related types

The `FunctionPIEDesc` is the additional information record which contains additional information that Argus ONE needs for each Function PIE. It contains information about the parameters types and names, the type of the return value and pointer to the function that does the calculation.

That function must be in the specific type as described in the section about the `PIEFunctionCall` below. It receives an array of pointers to the values of the parameters that you have requested in the `FunctionPIEDesc` record. You should do whatever calculation you need using those values and store the result in the storage area as pointed by the `result` pointer.

Below are several definitions of enumerations that are defined for your convenience. Use these enumerated types in the various fields as necessary:

`EFunctionPIEFlags` enumeration includes the different flags that you can use to customize the behavior of the Function PIE:

`kFunctionNeedsProject`
: If set, the field `neededProject` in `FunctionPIEDesc` must contain the name of a Project PIE, without which, the function will not be available (not used yet).

`kFunctionHasCategory`
: If set, the field `category` in `FunctionPIEDesc` mush contain the name of the category that the function belongs to. Category (like “Math”, “Conversion”,



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

etc.) is the name of a group of functions, that the function belongs to. This name is displayed in the left side list of the expression dialog (not used yet).

`kFunctionDisplaysDialogWhenDeclared`
Will allow function PIEs to display a dialog when declared by the Argus ONE user (not used yet).

`kFunctionIsHidden`
Will prevent the function from appearing in the Expression Dialog Box (the "Calculator" Dialog Box). Use this flag when declaring functions that are required internally by other PIEs in the library, and you don't want the user to access them (these functions **can** be used by the user, however, but they must be typed in explicitly).

### `PIEFunctionCall` pointer to function

Your Function PIE should supply Argus ONE with a pointer to the function that performs the needed calculation. This function should be in the exact form that Argus ONE expects it to be, otherwise a runtime error might occur when Argus ONE tries to call your function.

Your function should be defined as follows:

- 'C' version:
```
void MyFunction(const ANE_DOUBLE* refPtX, const ANE_DOUBLE* refPtY, ANE_INT16
numParams, const ANE_PTR* parameters, ANE_PTR funHandle, ANE_PTR reply)
```
- 'C++' version:
```
void MyFunction(const ANE_DOUBLE& refPtX, const ANE_DOUBLE& refPtY, ANE_INT16
numParams, const ANE_PTR* parameters, ANE_PTR funHandle, ANE_PTR reply)
```

`refPtX` Reference point x coordinate. Passed by reference.

`refPtY` Reference point y coordinate. Passed by reference.

`numParams` Number of parameters passed by value.

`parameters` Array of size `numParams` with pointers to the parameters values. The type of each value is in accordance with the type you have supplied in the `FunctionPIEInfo`.

`funHandle` The same handle that is passed in `functionHadnle` field of `FunctioPIEDesc` (not used yet).



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

`reply` A pointer to the address of the reply. The type of the pointer depends on the return type of the function. Your function must store the calculation result into that address.

### Constants

`FUNCTION_PIE_VERSION`
The version of the Function PIE interface. If you use this constant to initialize the `version` field of `FucntionDesc`, recompiling the source with future additions of the PIE interface will result PIE that is compatible with that latest version.

### FunctionPIEInfo structure

`version` Type: `ANE_INT32`
The version of the `FunctionPIEDesc` structure. This field is needed for Argus to be able to improve the API in the future and to add more fields to the descriptor. If you use the constant `FUNCTION_PIE_VERSION` to initialize this field, you can be sure that the version is correct when you recompile your PIE using a future addition of the PIE interface.

`functionFlags` Type: `EFunctionPIEFlags`
Flags that enable or disable features and/or functionality of the function PIE.

`name` Type: `ANE_STR`
The name of the function. In the current API version, this name should be the same as the name in the `ANEPIEDesc` that defines the function PIE.

`address` Type: `PIEFunctionCall`
Address of the function that does the calculation.See the section about `PIEFunctionCall` above. This field is the reason for function PIE.

`returnType` Type: `EPIENumberType`
The type of the function return value.

`numParams` Type: `ANE_INT16`
Number of mandatory parameters the function receives as input.

`numOptParams` Type: `ANE_INT16`
Maximum number of optional parameters the function can receive as input, in addition to the `numParams` mandatory parameters. For example if



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

numParams is 2 and numOptParams is 3, Argus ONE will let the user specify 2, 3, or 4 parameters for your function.

| | |
|---|---|
| paramNames | Type: ANE_STR*<br>An array of strings, terminated by a null pointer (**not a pointer to null string!** ), which contains the names of the parameters that the user sees in the Expression-Dialog when selecting the function name from the list. For example a function that converts from degree-minute-second notation to decimal degrees would have 3 parameters names:<br>paramNames = {“Degree”, “Minutes”, “Seconds”, 0}; |
| paramTypes | Type: EPIENumberType*<br>An array of types of the parameters. The array is terminated by zero. For the former example you should use 3 real parameters:<br>paramTypes = {kPIEFloat, kPIEFloat, kPIEFloat, 0}; |
| functionHandle | Type: ANE_PTR<br>A handle that can be later used by the function (not used yet). |
| category | Type: ANE_STR<br>The name of the category that function will appear in the expression dialog. This field is referenced only if the flags kFunctionHasCategory is set in the field functionFlags (not used yet). |
| neededProject | Type: ANE_STR<br>The name of a project that the function needs in order to be available. Use this field if your function is specific to a certain project and should not be called on any other document. This field is referenced only if the flag kFunctionNeedsProject is set in the field functionFlags (not used yet). |

### Implementing Simple Function

Below is a small sample code that defines a Function PIE that calculates the absolute value of it’s (single) parameter:

```
/*****************************************************************************\
*
*  MODULE:      AbsFunPIE.cp
*
*  PURPOSE:     To provide a Demo Function PIE template.
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
*
*  FUNCTIONS:
*
*  COMMENTS:
\*****************************************************************************/

/* for function PIE, include “FunctionPIE.h”, which includes also “ANEPIE.h” for you */
#include “FunctionPIE.h”

/* always remember to include <math.h> when calling mathematical functions */
/* that return doubles, or else ‘C’ assumes that the result is int.        */
#include <math.h>


/*===========================================================================
// PIE Function definition
//===========================================================================*/

/*---------------------------------------------------------------------------
// GPIEAbs:
//---------------------------------------------------------------------------
// Function definition:
// Define an abs function that (surprise!) return the absolute value of
// its parameter (of type double)
//
//---------------------------------------------------------------------------*/

static void
GPIEAbsMMFun( const ANE_DOUBLE* refPtX,          /* the x coord. of ref. point (not used) */
              const ANE_DOUBLE* refPtY,          /* the y coord. of ref. point (not used) */
              ANE_INT16 numParams,               /* num. of params (not used, should be 1) */
              const ANE_PTR* parameters,         /* array of (1) parameters */
              ANE_PTR funHandle,                 /* not used */
              ANE_PTR reply                      /* where the reply should return */ )
{
  /* extract the parameter for the function, which is known to be double */
  ANE_DOUBLE param1 = *(ANE_DOUBLE*)(parameters[0]);

  /* calculate the function */
  ANE_DOUBLE result = fabs(param1);

  /* return the result as a double*/
  *(ANE_DOUBLE*)reply = result;

} /* GRunPTCMMFun */
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
//================================================================
//PIE handling functions and declarations
//================================================================

struct ANEPIEDesc* gFunDesc[1];                 /* prepare a list of 1 PIE descriptor */

/* “PIE_Abs” function part */

struct ANEPIEDesc gPIEAbsDesc;                  /* PIE defcriptor */
struct FunctionPIEDesc gPIEAbsFDesc;            /* Function PIE information record */
char* gpnNumber[2] = {“Number”, 0};             /* list of one parameter name */
enum EPIENumberTypegOneFloatTypes[2]            /* list of parameters types for the function */
             = {kPIEFloat, (EPIENumberType)0};  /* one float parameters */
void
GetANEFunctions(ANE_INT32* numNames, struct ANEPIEDesc*** descriptors)
{
  *numNames = 0;

  /* prepare Function PIE information record for “PIE_Abs” function */

  gPIEAbsFDesc.version = FUNCTION_PIE_VERSION;                /* use constant here */
  gPIEAbsFDesc.functionFalgs = (EFunctionPIEFlags)0;          /* no special features */
  gPIEAbsFDesc.name = “PIE_Abs”;                              /* name of function */
  gPIEAbsFDesc.address = GPIEAbsMMFun;                        /* function address */
  gPIEAbsFDesc.type = kPIEFloat;                              /* return value type */
  gPIEAbsFDesc.numParams = 1;                                 /* number of parameters */
  gPIEAbsFDesc.numOptParams = 0;                              /* number of optional parameters */
  gPIEAbsFDesc.paramNames = gpnNumber;                        /* ptr to parameter names list */
  gPIEAbsFDesc.paramTypes = gOneFloatTypes;                   /* ptr to parameters types list */

  /* prepare Argus ONE PIE descriptor for “PIE_Abs” function  */

  gPIEAbsDesc.version = ANE_PIE_VERSION;                      /* use constant here */
  gPIEAbsDesc.name = “PIE_Abs”;                               /* name of PIE */
  gPIEAbsDesc.type = kFunctionPIE;                            /* PIE Type: PIE function */
  gPIEAbsDesc.descriptor = &gPIEAbsFDesc;                     /* pointer to descriptor */

  gFunDesc[(*numNames)++] = &gPIEAbsDesc;                     /* add descriptor to list */

  *descriptors = gFunDesc;                                    /* return array of descriptors */
}
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

Future examples will include:

- Implementing functions with variable number of parameters
- Implementing functions that receive Real, Int, String
- Implementing several function within a single implementation

## Export Template PIEs

`PIEExportGetTemplate` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_STR* returnTemplate)
```

| | |
|---|---|
| `aneHandle` | a handle to be used when calling ANE_Xxx functions from this callback. This handle is valid only in this function call scope. |
| `returnTemplate` | a pointer where the template (of type ANE_STR) should be stored. Make sure the memory that contains the template is no automatic (on the stack). The memory can be either static or on the heap (malloc, new). |

`PIEExportCallBack` **callback definition:**

```
void Xxxx(ANE_PTR aneHandle)
```

| | |
|---|---|
| `aneHandle` | a handle that is used then calling ANE_Xxx functions from this callback. This handle is valid only in this function call scope. |

`EExportTemplateFlags` **enumeration**

| | |
|---|---|
| `kExportDontShowParamDialog` | if this flag is set, Argus ONE will not display the standard export parameters dialog |
| `kExportDontShowFileDialog` | if this flag is set, Argus ONE will not display the file dialog (the dialog where the user can select the name of file to export to.) |
| `kExportCallPreExport` | if this flag is set, the function whose address is stored in preExportProc field of ExportTemplatePIEDesc structure, is called before the export process starts (not used yet). |



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

`kExportCallPostExport`
if this flag is set, the function whose address is stored in postExportProc field of ExportTemplatePIEDesc structure, is called after the export process ends (not used yet).

`kExportDontGetTemplate`
if this flag is set, the function, whose address is stored in getTempalteProc field of ExportTemplatePIEDesc structure, is NOT called in order to get the export template. This means that if the PIE exports the layer, it doesn't use the export template to do this job (not used yet).

`kExportNeedsProject` if this flag is set, the field neededProject of ExportTemplateDesc structure contains the name of a project, without which, the Export Template PIE will not add a menu item to the export sub menu (not used yet).

### `ExportTemplatePIEDesc` structure

version — Type: ANE_INT32
the version of Export Template PIE interface. If you use `EXPORT_TEMPLATE_PIE_VERSION` constant, then whenever you recompile the Export Template PIE, the latest version is set to this field.

name — Type: ANE_STR
the name of the Export Template PIE. In this version, this name should be the same as name field in the ANEPIEDesc structure that refer to the Export Template PIE.

`exportType` — Type: `EPIELayerType`
mask of layer types that the Export Template PIE can handle. The name of the Export Template PIE will appear in the Export submenu of every layer whose type is masked by this field.

exportFlags — Type: EExportTemplateFlags
flags the enable and disable special features of the Export Template PIE. See description of EExportTemplateFlags for available features.

`getTemplateProc` — Type: `PIEExportGetTemplate`
function to be called in order to get an export template to be executed for the layer from which the menu item was called. This function is called only if flag kExportDontGetTemplate is NOT set in exportFlags field.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

| | |
|---|---|
| `preExportProc` | Type: `PIEExportCallBack`<br>function to be called before the export process starts. This function is called only if flag kExportCallPreExport is set in exportFlags field (no used yet). |
| `postExportProc` | Type: `PIEExportCallBack`<br>function to be called after the export process ends. This function is called only if flag kExportCallPostExport is set in exportFlags field (no used yet). |
| `neededProject` | Type: `ANE_STR`<br>name of the project that the Export Template PIE requires. If the flag kExportRequiresProject is set in the exportFlags field, then the menu item for this export template will be added only to documents that were created by neededProject (no used yet). |

### Using Function PIE to improve Export Template PIE performance

It is possible to improve performance of some actions of the Export Template PIE using FunctionPIEs. You can use the `kFunctionIsHidden` flag to hide these functions from the user.

To call Function PIEs as procedures from an Export Template in general, and Export Template PIEs in particular use the “Evaluate Expression” command:

```
Evaluate Expression: SAMPLE_FUNCTION_PIE()
```

### Implementing Simple Export Template PIE

7 steps of developing Export Template Extension:

- Create a working export template
- Define a function that return the template (see GSimpleExportTemplate)
- Define ANEPIEDesc and ExportTemplateDesc structures
- Define GetANEFunctions function
- Fill the structures you declared in step 3
- Define an array of ANEPIEDesc pointers (gPIEDesc)
- Add the address of the structure you created in step 3 to the array.

```
/*****************************************************************************\
*
*  MODULE:      simple-export.c
*
\*****************************************************************************/
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
#include “ExportTemplatePIE.h”

/*-------------------------------------------------------------------------
// GSimpleExportTemplate:
//-------------------------------------------------------------------------*/

void GSimpleExportTemplate(ANE_PTR Handle, ANE_STR* returnTemplate)
{
  static char template[] =
    “Redirect output to: $BaseName$\n”
    “Loop for: Elements\n”
    “Start a new line\n”
    “Export expression: \”Triangle \”\n”
    “Export expression: NthNodeX(1); [F8.2]\n”
    “Export expression: NthNodeY(1); [F8.2]\n”
    “Export expression: NthNodeX(2); [F8.2]\n”
    “Export expression: NthNodeY(2); [F8.2]\n”
    “Export expression: NthNodeX(3); [F8.2]\n”
    “Export expression: NthNodeY(3); [F8.2]\n”
    “End line\n”
    “End loop\n”
    “End file\n\n”;

  *returnTemplate = template;
} /* GSimpleExportTemplate */

static struct ExportTemplatePIEDesc gSimpleExportTemplateTDesc;
static struct ANEPIEDesc gSimpleExportTemplateDesc;

static struct ANEPIEDesc* gPIEDesc[1];

void
GetANEFunctions(long* numNames, struct ANEPIEDesc*** descriptors)
{

  *numNames = 0;
  gSimpleExportTemplateTDesc.version = EXPORT_TEMPLATE_PIE_VERSION;
  gSimpleExportTemplateTDesc.name = “Simple Export PIE”;
  gSimpleExportTemplateTDesc.exportType = kPIETriMeshLayer;
  gSimpleExportTemplateTDesc.exportFlags = kExportDontShowParamDialog;
  gSimpleExportTemplateTDesc.getTemplateProc = GSimpleExportTemplate;

  gSimpleExportTemplateDesc.name = “Simple Export PIE”;
  gSimpleExportTemplateDesc.type = kExportTemplatePIE;
  gSimpleExportTemplateDesc.descriptor = &gSimpleExportTemplateTDesc;
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
  gPIEDesc[*numNames] = &gSimpleExportTemplateDesc;

  (*numNames)++;


  *descriptors = gPIEDesc;
}
```

## Import Extension

`ImportPIEInfo` **structure**

| | |
|---|---|
| `version` | Type: `ANE_INT32`<br>the version of the Import PIE. If you use the constant IMPORT_PIE_VERSION, then whenever you recompile the Import PIE the latest version number is stored in this field. |
| `name` | Type: `ANE_STR`<br>the name of the Import PIE. In this version of Import PIE, this name should be the same as the name field of the ANEPIEDesc structure that refers to the ImportPIEInfo structure |
| `importFlags` | Type: `EImportPIEFlags`<br>flags that enable or disable special features of the Import PIE. See below for description of EImportPIEFlags values. |
| `toLayerTypes` | Type: `EPIELayerType`<br>mask of layer types that this Import PIE can import to. |
| `fromLayerTypes` | Type: `EPIELayerType`<br>mask of layer types that the Import PIE can import from. This field is ignored unless kImportFromLayer is set in the importFlags (not used yet). |
| `doImportProc` | Type: `PIEImportProc`<br>the procedure that is called when the user select the menu item from the Import sub menu. See description of PIEImportProc for more information. |
| `neededProject` | Type: `ANE_STR`<br>if the flag `kImportNeedsProject` is set, then the Import PIE name is added |



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

to the Import submenu only if the name of the project that created the document equals to this field (not used yet).

`EImportPIEFlags` **enumeration**

| | |
|---|---|
| `kImprotFromFile` | if this flag is set, then the Import File Dialog is displayed, to let the user choose the file to import from. The function pointed by doImportProc receives the name of the file chosen. |
| `kImportFromLayer` | if this flag is set, then a dialog with a list of layers that match the field fromLayerTypes is displayed. A handle to the layer selected by the user is passed to the function pointed by doImportProc.(not used yet) |
| `kImportNeedsProject` | if this flag is set, then only documents that were created by a Project PIE whose name equals to the neededProject field can use the Import PIE. |

| | |
|---|---|
| `IMPORT_PIE_VERSION` | Type: constant<br>a constant to be used to initialize the version field of ImportPIEDesc structure. If you use this constant to initialize the field, each time you recompile the Import PIE you automatically use the latest version of the Import PIE interface. |

`PIEImportProc` **callback definition:**

```
void Xxxx(ANE_PTR aneHandle, cosnt ANE_STR fileName, ANE_PTR layerHandle)
```

| | |
|---|---|
| `aneHandle` | a handle to be passed to each ANE_Xxxx function called by this callback. This handle is valid only in the scope of the call. |
| `fileName` | if `kImprotFromFile` is set, then this parameter contains the full name (path and file name) of the file chosen by the user. |
| `layerHandle` | if `kImportFromLayer` is set, then this parameter contains a handle of the layer the user chose to import from (not used yet). |

**Related Argus ONE callback functions:**

```
ANE_ImportTextToLayer(ANE_PTR aneHandle, ANE_STR buf);
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

| | |
|---|---|
| aneHandle | the handle that was passed as a parameter to the doImportProc |
| buf | a string that contains the text to be imported to the layer. The format of the string is the same as the format of a file imported to the layer using the Import Text menu item. |

### Importing from external file

```
/*****************************************************************************\
*
*  MODULE:      simple_import.c
*
\*****************************************************************************/

#include “ImportPIE.h”
#include “ANECB.h”
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>


/*---------------------------------------------------------------------------
// sReserve
//---------------------------------------------------------------------------*/

static sReserve(char** buf, int* bufLen, int minLength)
{
  int newLen;

  if (*bufLen > minLength)
    return;

  newLen = *bufLen*1.5;
  if (newLen <= minLength)
    newLen = minLength + 32;

  if (*buf)
    *buf = realloc(*buf, newLen);
  else
    *buf = malloc(newLen);

  *bufLen = newLen;
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
}

/*---------------------------------------------------------------------------
// GDrawerImportProc
//---------------------------------------------------------------------------*/

static void GSimpleImportProc(ANE_PTR aneHandle, ANE_STR fileName, ANE_PTR layerHandle)
{
/*************************************************************************
*************************************************************************
**
** This function is where 'Import' extension works.
**
**  The input parameters are:
**  aneHandler - a handle that you pass to Argus Open Numerical Environments API
**  fileName - the file name of the file selected by the user (if you asked one)
**  layerHandle - (for future use) the handle of the layer selected by the user
**            this handle will be used to get information from this layer
**
**  This function is very simple:
**  It defines the string to be imported (a single well at position (10,10))
**  and calls the function ANE_ImportTextToLayer(aneHandle, imp) where
**  aneHandle is the first parameter transferred to this function
**  imp is the import string
**  It also printf the fileName and the import string to stdout, for debugging
**  purposes.
**
**  Where to go from here?
**
**    The most trivial path is to use fileName to open the file, read information
**    from it, and creating import string using the information from the file
**
**  Many other options are available, including displaying a dialog and using
**  the parameter from the dialog to create information, or even creating random
**  information, without using any data, neither from user, nor from file.
**
*************************************************************************
*************************************************************************/


  char* buf = 0;
  int bufLen = 0;
  int loc = 0;
  FILE* f = fopen(fileName, "r");
  double centerx, centery, radius;
  int numPoints = 20;
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
  int i;
  double angle;
  int n;

  fscanf(f, "%lf %lf %lf", &centerx, &centery, &radius);

  sReserve(&buf, &bufLen, loc+17);
  sprintf(buf+loc,"%5d %10f\n%n", numPoints+1, M_PI*radius*radius, &n);
  loc+= n;

  for (i = 0; i <= numPoints; i++)
  {
    angle = 2.0*M_PI*(double)(i%numPoints)/(double)numPoints;

    sReserve(&buf, &bufLen, loc+11);
    sprintf(buf+loc,"%10lf %n", centerx+radius*cos(angle), &n);
    loc+= n;

    sReserve(&buf, &bufLen, loc+11);
    sprintf(buf+loc,"%10lf\n%n", centery+radius*sin(angle), &n);
    loc+= n;
  }
  buf[loc] = '\0';

  ANE_ImportTextToLayer(aneHandle, buf);

  free(buf);

} /* GDrawerImportProc */


/*================================================================
//PIE handling functions
//================================================================*/


struct ImportPIEDesc  gSimpleImportIDesc;
struct ANEPIEDesc     gSimpleImportDesc;

struct ANEPIEDesc*    gFunDesc[20];


void
GetANEFunctions(ANE_INT32* numNames, struct ANEPIEDesc*** descriptors)
{
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
  *numNames = 0;


  gSimpleImportIDesc.version = IMPORT_PIE_VERSION;
  gSimpleImportIDesc.name = “SimpleImport”;
  gSimpleImportIDesc.importFlags = kImportFromFile;
  gSimpleImportIDesc.toLayerTypes = kPIEInformationLayer;
  gSimpleImportIDesc.fromLayerTypes = kPIETriMeshLayer | kPIEInformationLayer;
  gSimpleImportIDesc.doImportProc = GSimpleImportProc;

  gSimpleImportDesc.name = “Simple Import...”;
  gSimpleImportDesc.type = kImportPIE;
  gSimpleImportDesc.descriptor = &gSimpleImportIDesc;

  gFunDesc[*numNames] = &gSimpleImportDesc;

  (*numNames)++;
  *descriptors = gFunDesc;
}
```

Importing from another layer

Using internal algorithms

generating random data

## Project PIEs

### ProjectPIEDesc structure

`version` Type: `ANE_INT32`
the version of the Project PIE interfaces. If you use `PROJECT_PIE_VERSION` to initialize this field, every time you recompile the Project PIE, you automatically use the latest version of the API.

`name` Type: `ANE_STR`
the name of the project. In the current version of the interface, this name should be the same as the name field of the `ANEPIEDesc` structure that refers to this structure.

`projectFlags` Type: `EProjectPIEFlags`
flags the enable or disable special features of the Project PIE. See



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

`EProjectPIEFlags` for a more complete description of the actions controlled by the flags.

`createNewProc` Type: `PIEProjectNew`
a pointer to a function called when a new project is created. See the description of PIEProjectNew call back for the parameters that are passed to the PIE.

`editProjectProc` Type: `PIEProjectEdit`
a pointer to a function called when the “Edit Project Info...” menu item is selected by the user. The menu item is available only if the project has the flag `kProjectCanEdit` set in the projectFlags field. See below the description of PIEProjectEdit for more information about the parameters that are passed to the function.

`cleanProjectProc` Type: `PIEProjectClean`
a pointer to a function called when a document that was created by the project is about to be closed. This function is called only if the flag `kProjectShouldClean` is set in the projectFlags field. See below the description of PIEProjectClean for more info about the parameters that passed to the function.

`saveProc` Type: `PIEProjectSave`
a pointer to a function called when the document that was created by the project is saved. This function is called only if the flag `kProjectShouldSave` is set. See description of PIEProjectSave for more information about the parameters that are passed to the function.

`loadProc` Type: `PIEProjectLoad`
a pointer to a function called when the document that was created by the project is loaded. This function is called only if the flag `kProjectShouldSave` is set. See description of PIEProjectLoad for more information about the parameters that are passed to the function.

### `EProjectPieFlags` enumeration

`kProejctDisplaysDialog` (not used yet)

`kProjectShouldSave` if this flag is set, the functions pointed by saveProc and LoadProc are called when a project created by the project is saved or loaded respectively.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

`kProjectCanEdit` if the flag is set, a menu item “Edit Project Info...” is added to the edit menu. When this menu item is selected, the function pointed by editProjectProc is called.

`kProjectShouldClean` if this flags is set, the function pointed by cleanProjectProc is called before the document is closed. This allows the project to free dynamic memory allocated when the project created the document, or loaded it from disk.

`PROJECT_PIE_VERSION` Type: constant
a constant that should be used to initialize the version field of ProjectPIEDesc. If you use this constant, then every time you recompile the Project PIE, you automatically use the latest version of the interface.

`PIEProjectNew` **callback definition**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR* rPIEHandle, ANE_STR* returnLayerTemplate)
```

`aneHandle` a handle that should be passed to all ANE_Xxx function called from this callback. This handle is valid only in this callback scope.

`rPIEHandle` a pointer to a place where the Project PIE can store a handle that will be passed to its callback functions.

`retrunLayerTemplate` a pointer to a place where the Project PIE should save the layer structure of the document it is about to create. The memory that contain this structure should not be automatic (that it, on the stack), but rather static or dynamic (in the heap).

`PIEProjectEdit` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR PIEHandle)
```

`aneHandle` a hanle that should be passed to all ANE_Xxx function called from this callback. This handle is valid only in this callback scope.

`PIEHandle` the handle that was returned by the newProc in rPIEHandle.

`PIEProjectClean` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR PIEHandle)
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

| | |
|---|---|
| `aneHandle` | a handle that should be passed to all ANE_Xxx function called from this callback. This handle is valid only in this callback scope. |
| `PIEHandle` | the handle that was returned by the newProc in rPIEHandle. |

`PIEProjectSave` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR PIEHandle, ANE_STR* rSaveInfo)
```

| | |
|---|---|
| `aneHandle` | a hanle that should be passed to all ANE_Xxx function called from this callback. This handle is valid only in this callback scope. |
| `PIEHandle` | the handle that was returned by the newProc in rPIEHandle. |
| `rSaveInfo` | a pointer to a place where a null terminated string that contains the information of the Project PIE to be saved. Make sure that the memory that contains this information is not automatic (that is, on the stack) but static or dynamic (on the heap) |

`PIEProjectLoad` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR* rPIEHandle, ANE_STR laodInfo)
```

| | |
|---|---|
| `aneHandle` | a hanle that should be passed to all ANE_Xxx function called from this callback. This handle is valid only in this callback scope. |
| `rPIEHandle` | a pointer where the memory of the project handle should be saved. The project handle has the same information and meanings as the one return by the newProjectProc, since this handle is the one passed to all other callbacks. |
| `loadInfo` | a null terminated string that contains the information saved by saveProc. |

### Simple Project PIEs

```
/*****************************************************************************\
*
*  MODULE:      simple-project.c
*
\*****************************************************************************/

#include “ProjectPIE.h”
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
/*---------------------------------------------------------------------------
// GSimpleProject:
//---------------------------------------------------------------------------*/

static
void GSimpleExportNew(ANE_PTR Handle, ANE_PTR* rPIEHandle, ANE_STR* rLayerTemplate)
{
  static char layerStructure[] =
    "Layer:\n"
    "{\n"
    "Name: \"Simple Domain\"\n"
    "Units: \"Density\"\n"
    "Type: \"Domain\"\n"
    "Visible: Yes\n"
    "Interpretation Method: Nearest\n"
    "\n"
    "Parameter: \n"
    "{\n"
    "Name: \"Density\"\n"
    "Units: \"Units\"\n"
    "Value Type: Real\n"
    "Value: \"0\"\n"
    "Parameter Type: Layer\n"
    "}\n"
    "}\n"
    "Layer:\n"
    "{\n"
    "Name: \"Simple Density\"\n"
    "Units: \"Units\"\n"
    "Type: \"Information\"\n"
    "Visible: Yes\n"
    "Interpretation Method: Nearest\n"
    "\n"
    "Parameter: \n"
    "{\n"
    "Name: \"Simple Density\"\n"
    "Units: \"Units\"\n"
    "Value Type: Real\n"
    "Value: \"0\"\n"
    "Parameter Type: Layer\n"
    "}\n"
    "}\n"
    "\n"
    "\n"
    "Layer:\n"
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
    "{\n"
    "Name: \"SimpleGrid\"\n"
    "Units: \"Units\"\n"
    "Type: \"Grid\"\n"
    "Visible: Yes\n"
    "Domain Layer: \"Simple Domain\"\n"
    "Density Layer: \"Simple Density\"\n"
    "\n"
    "Template: \n"
    "{\n"
    "Line: \"File: $BaseName$\"\n"
    "Line: \"\\tLine\"\n"
    "Line: \"\\tExpr: NumRows(); [I8]\"\n"
    "Line: \"\\tExpr: NumColumns(); [I8]\"\n"
    "Line: \"\\tExpr: NumBlockParameters()+1 [I8]\"\n"
    "Line: \"\\tEnd line\"\n"
    "Line: \"\\tLoop: Rows\"\n"
    "Line: \"\\tLine\"\n"
    "Line: \"\\tExpr: NthRowPos($Row$) [F8.2]\"\n"
    "Line: \"\\tEnd line\"\n"
    "Line: \"\\tEnd loop\"\n"
    "Line: \"\\tLoop: Columns\"\n"
    "Line: \"\\tLine\"\n"
    "Line: \"\\tExpr: NthColumnPos($Column$) [F8.2]\"\n"
    "Line: \"\\tEnd line\"\n"
    "Line: \"\\tEnd loop\"\n"
    "Line: \"\\tMatrix: BlockIsActive() [I1]\"\n"
    "Line: \"\\tLoop: Block Parameters\"\n"
    "Line: \"\\tLine\"\n"
    "Line: \"\\tEnd line\"\n"
    "Line: \"\\tLine\"\n"
    "Line: \"\\tExpr: \\\"# $Parameter$\\\"\"\n"
    "Line: \"\\tEnd line\"\n"
    "Line: \"\\tMatrix: $Parameter$ [F8.2]\"\n"
    "Line: \"\\tEnd loop\"\n"
    "Line: \"\\tEnd file\"\n"
    "Line: \"\"\n"
    "}\n"
    "}\n\n";

  *rLayerTemplate = layerStructure;

} /* GSimpleExportTemplate */

static struct ProjectPIEDesc gSimpleProjectPDesc;
static struct ANEPIEDesc gSimpleProjectDesc;
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
static struct ANEPIEDesc* gPIEDesc[20];

void
GetANEFunctions(long* numNames, struct ANEPIEDesc*** descriptors)
{
  *numNames = 0;

  gSimpleProjectPDesc.version = PROJECT_PIE_VERSION;
  gSimpleProjectPDesc.name = “Simple Project”;
  gSimpleProjectPDesc.projectFlags = (enum EProjectPIEFlags)0;
  gSimpleProjectPDesc.createNewProc = GSimpleExportNew;
  gSimpleProjectPDesc.editProjectProc = 0;
  gSimpleProjectPDesc.cleanProjectProc = 0;
  gSimpleProjectPDesc.saveProc = 0;
  gSimpleProjectPDesc.loadProc = 0;

  gSimpleProjectDesc.name = “Simple Project”;
  gSimpleProjectDesc.type = kProjectPIE;
  gSimpleProjectDesc.descriptor = &gSimpleProjectPDesc;

  gPIEDesc[*numNames] = &gSimpleProjectDesc;

  (*numNames)++;

  *descriptors = gPIEDesc;
}
```

### Integrating Template Extension and Project Extension

Not documented yet.

## Interpolation PIEs

Interpolation PIEs add new methods for interpolating data points or contour vertices. The PIE supplies a call-back function which will be called by Argus ONE to calculate the layer values at given *x*, *y* coordinates. The PIE can also use a block of information that reflects all the data points in the layer. In order to prepare such a block the PIE should supply Argus ONE with a preparation call-back function. This function receives all the data points in the layer and their values, it should then calculate any appropriate information and supply Argus ONE with a pointer to that information block. This pointer will be passed to the calculation call-back function in all consecutive interpolation calls.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

`PIEInterpolationPreProc` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR* rPIEHandle, ANE_INT32 numPoints, ANE_DOUBLE* xCoords,
ANE_DOUBLE* yCoords, ANE_DOUBLE* values)
```

| | |
|---|---|
| `aneHandle` | a handle to be used when calling ANE_Xxx functions from this callback. This handle is valid only in this function call scope. |
| `rPIEHandle` | put here a pointer to the PIE's memory block. This pointer will be passed then to the interpolation call-back function. |
| `numPoints` | number of data points (or contour vertices) in the layer. |
| `xCoords` | an array of `numPoints` values, representing the *x* coordinates of the data points. |
| `yCoords` | an array of `numPoints` values, representing the *y* coordinates of the data points. |
| `values` | an array of `numPoints` values, representing the values of the data points. |

`PIEInterpolationEvalProc` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR pieHandle, ANE_DOUBLE x, ANE_DOUBLE y, ANE_DOUBLE* rResult)
```

| | |
|---|---|
| `aneHandle` | a handle to be used when calling ANE_Xxx functions from this callback. This handle is valid only in this function call scope. |
| `pieHandle` | a pointer to the PIE's memory block (which was given by `rPIEHandle` previously). |
| `x` | *x* coordinate of the required interpolation point. |
| `y` | *y* coordinate of the required interpolation point. |
| `rResult` | put the result of the calculation here. |

`PIEInterpolationClean` **callback definition:**

```
void Xxx(ANE_PTR aneHandle, ANE_PTR pieHandle)
```

| | |
|---|---|
| `aneHandle` | a handle to be used when calling ANE_Xxx functions from this callback. This handle is valid only in this function call scope. |
| `pieHandle` | a pointer to the PIE's memory block, should be cleaned up. |



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

`EInterpolationFlags` **enumeration**

`kInterpolationIncremental`
Not implemented yet

`kInterpolationCallPre`
if this flag is set, Argus ONE will call the call-back function `PIEInterpolationPreProc.`

`kInterpolationClean`
if this flag is set, Argus ONE will call the call-back function `PIEInterpolationClean.`

`ExportTemplatePIEDesc` **structure**

`version` Type: ANE_INT32
the version of Interpolation PIE interface. If you use `INTERPOLATION_PIE_VERSION` constant, then whenever you recompile the Interpolation PIE, the latest version is set to this field.

`name` Type: ANE_STR
the name of the Interpolation PIE. In this version, this name should be the same as name field in the ANEPIEDesc structure that refer to the Interpolation PIE.

`interpolationFlags` Type: `EInterpolationFlags`
flags the enable and disable special features of the Interpolation PIE. See description of `EInterpolationFlags` for available features.

`preProc` Type: `PIEInterpolationPreProc`
pointer to the function to be called, before the interpolation process, in order to calculate the PIE's internal information as a preparation to the interpolation phase. This function is called only if the flag `kInterpolationCallPre` is set in the `interpolationFlags` field.

`evalProc` Type: `PIEInterpolationEvalProc`
pointer to the function to be called to interpolate the layer's value at a given point.

`cleanProc` Type: `PIEInterpolationClean`
pointer to the function to be called, after the interpolation process, in order to



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

clean up the PIE's memory block. This function is called only if the flag `kInterpolationClean` is set in the `interpolationFlags` field.

### Implementing Simple Interpolation PIE

Below is a small sample code that defines a new interpolation method titled "SimpleInterpolate" that only returns the value of the closest data point:

```
/*****************************************************************************\
*
*  MODULE:      Simple-Interpolate.
*
\*****************************************************************************/

#include "Simple-Interpolate.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#define MAIN_MODULE

#ifdef MAC
#pragma lib_export on
extern "C" void GetANEFunctions(ANE_INT32* numNames, struct ANEPIEDesc*** descriptors);
#pragma lib_export off
#endif

/*---------------------------------------------------------------------------
// GSimplePreInterpolateProc
//---------------------------------------------------------------------------*/

static void GSimplePreInterpolateProc(ANE_PTR aneHandle, ANE_PTR* rMyHandle,ANE_INT32 numPoints,
ANE_DOUBLE* x, ANE_DOUBLE* y, ANE_DOUBLE* values)
{
/*****************************************************************************
*****************************************************************************
**
** This function is where 'Pre-Interploate' extension works, that is where the preparation of
** interpolation happens.
**
** The input parameters are:
**  aneHandler - a handle that you pass to Argus Open Numerical Environments API
**  numPoints  - number of given data points to interpolate from
**  x          - an array of the x coordinates of the given data points
**  y          - an array of the y coordinates of the given data points
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
**  values     - an array of values of the interpolation points
**
** The output parameters are:
**  rMyHandle  - a handle to be passed by ArgsuNE to the Interpolate procedure
**               ANE to the Clean procedure
*****************************************************************************
*****************************************************************************/
  int i;
  double* data = (double*)malloc((1+3*numPoints)*sizeof(double));

  data[0] = numPoints;
  for (i = 0; i < numPoints; i++)
  {
    data[i*3+1] = x[i];
    data[i*3+2] = y[i];
    data[i*3+3] = values[i];
  }

  *rMyHandle = data;
} /* GSimplePreInterpolateProc */

/*---------------------------------------------------------------------------
// GSimpleInterpolateProc
//-------------------------------------------------------------------------*/

static void GSimpleInterpolateProc(ANE_PTR aneHandle, ANE_PTR myHandle, ANE_DOUBLE x, ANE_DOUBLE y,
ANE_DOUBLE* result)
{
/*****************************************************************************
*****************************************************************************
**
** This function is where 'Interpolate' extension works.
**
** The input parameters are:
**  aneHandler - a handle that you pass to Argus Open Numerical Environments API
**  myHandle   - a handle returned by the PreInterpolate function
**  x          - the x coordinate of the point to be interpolated
**  y          - the y coordinate of the point to be interpolated
**
** The output parameters are:
**  result - a pointer to where the result of interpolation should be placed.
*****************************************************************************
*****************************************************************************/
  int i;
  double minDistanceSqr = -1.0;
  double distanceSqr;
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
  double curX, curY;
  double lastVal;
  double* info = (double*)myHandle;
  int numPoints = info[0];

  for (i = 0; i < numPoints; i++)
  {
    curX = info[i*3+1];
    curY = info[i*3+2];
    distanceSqr = (x-curX)*(x-curX)+(y-curY)*(y-curY);
    if (minDistanceSqr < 0.0 || minDistanceSqr > distanceSqr)
    {
      minDistanceSqr = distanceSqr;
      lastVal = info[i*3+3];
    }
  }

  *result = lastVal;
} /* GSimpleInterpolateProc */

/*--------------------------------------------------------------------------
// GSimpleInterpolateCleanProc
//--------------------------------------------------------------------------*/

static void GSimpleInterpolateCleanProc(ANE_PTR aneHandle, ANE_PTR myHandle)
{
/*************************************************************************
*************************************************************************
**
** This function should clean any memory allocated by the PreInterpolate function
**
** The input parameters are:
**           aneHandler - a handle that you pass to Argus Open Numerical Environments API
**           myHandle - a handle returned by the PreInterpolate function
*************************************************************************
*************************************************************************/
  free(myHandle);
} /* GSimpleInterpolateCleanProc */

/*=========================================================================
//           PIE handling functions
//=========================================================================*/

#pragma mark PIE handling functions

  struct InterpolationPIEDesc gSimpleInterpolateIDesc;
```



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

```
  struct ANEPIEDesc gSimpleInterpolateDesc;

  struct ANEPIEDesc* gFunDesc[20];

void
GetANEFunctions(ANE_INT32* numNames, struct ANEPIEDesc*** descriptors)
{
  *numNames = 0;

  gSimpleInterpolateIDesc.version = INTERPOLATION_PIE_VERSION;
  gSimpleInterpolateIDesc.name = "SimpleInterpolate";
  gSimpleInterpolateIDesc.interpolationFlags = (enum
EInterpolationFlags)(kInterpolationCallPre|kInterpolationShouldClean);
  gSimpleInterpolateIDesc.preProc = GSimplePreInterpolateProc;
  gSimpleInterpolateIDesc.evalProc = GSimpleInterpolateProc;
  gSimpleInterpolateIDesc.cleanProc = GSimpleInterpolateCleanProc;

  gSimpleInterpolateDesc.name = "SimpleInterpolate";
  gSimpleInterpolateDesc.type = kInterpolationPIE;
  gSimpleInterpolateDesc.descriptor = &gSimpleInterpolateIDesc;

  gFunDesc[*numNames] = &gSimpleInterpolateDesc;

  (*numNames)++;

  *descriptors = gFunDesc;
}
```

## Multi platform support

### Writing platform independent PIEs

All the generic examples in the above sections are platform independent, that is, you can compile them as is on any platform on which Argus ONE-PIEs are supported.

Points to notice when writing platform independent code

Always use type defined in `ANEPIE.h`:

- `ANE_INT16` for 16 bit integer
- `ANE_INT32` for 32 bit integer
- `ANE_STR` for null terminated strings
- `ANE_PTR` for void pointers and handles
- `ANE_DOUBLE` for floating point numbers



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

**Don’t use any compiler specific extensions**

ANE_GUI API

## Platform specific support - Visual C++ and Argus ONE-PIEs

### Building generic Argus ONE-PIE project in VC++

- Select new...
- Select Project Workspace
- Select Dynamic-Link Library
- Fill the AppWizard dialog with the relevant information
- Select Insert->Files into Project...
- Add the generic file to the project
- Crate a new text file
- Name it <ProjectName>.def
- Add the following lines to the file
  LIBRARY “<libName>”
  DESCRIPTION “<description of the pie>”
  EXPORTS
  GetANEFunctions @1
- Compile the dll, and the ANE-PIE is ready
- If you use ANE_Xxx callbacks - insert the file “..\ANECB.h” to the project
- If you need to use MFC in your PIE:
  ◊ Select new...
  ◊ Select Project Workspace
  ◊ Select MFC DLL
  ◊ Fill the AppWizard dialog with the relevant information
  ◊ Select Insert->Files into Project...
  ◊ Add the generic file into the project
  ◊ Add the line ‘GetANEFUnctions @1’ to the .def file of the project

### Sample SGI extension

Not documented yet.

### Sample CodeWarrior extension

Not documented yet.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800

### Using special features of Visual C++

Not documented yet.

### Using special features of CodeWarrior

Not documented yet.

### Using special feature of Motif and GL

Not documented yet.

## How to...

### Add special conversion function

Not documented yet.

### Support model solver code

Not documented yet.

### Import legacy data files

Not documented yet.

### Import model output files

Not documented yet.

### Support special post processing

Not documented yet.



350 5th Ave., Suite 2819, New York, NY 10118 Tel: (212) 563-8600 Fax: (212) 594-0800