Foreign Objects and Foreign Processes

Release v3.5

June 2012

Foreign Objects and Foreign Processes

Release v3.5
June 2012

Process Systems Enterprise Limited
6th Floor East
26-28 Hammersmith Grove
London W6 7HA
United Kingdom
Tel: +44 20 85630888
Fax: +44 20 85630999
WWW: http://www.psenterprise.com

Acknowledgements

ModelBuilder uses the following third party free-software packages. The distribution and use of these libraries is governed by their respective licenses which can be found in full in the distribution. Where required, the source code will made available upon request. Please contact support.gPROMS@psenterprise.com in such a case.

Many thanks to the developers of these great products!

Table 1. Third party free-software packages

Software/Copyright	**Website**	**License**
ANTLR	http://www.antlr2.org/	Public Domain
Batik	http://xmlgraphics.apache.org/batik/	Apache v2.0
Copyright © 1999-2007 The Apache Software Foundation.		
BLAS	http://www.netlib.org/blas	BSD Style
Copyright © 1992-2009 The University of Tennessee.		
Boost	http://www.boost.org/	Boost
Copyright © 1999-2007 The Apache Software Foundation.		
Castor	http://www.castor.org/	Apache v2.0
Copyright © 2004-2005 Werner Guttmann		
Commons CLI	http://commons.apache.org/cli/	Apache v2.0
Copyright © 2002-2004 The Apache Software Foundation.		
Commons Collections	http://commons.apache.org/collections/	Apache v2.0
Copyright © 2002-2004 The Apache Software Foundation.		
Commons Lang	http://commons.apache.org/lang/	Apache v2.0
Copyright © 1999-2008 The Apache Software Foundation.		
Commons Logging	http://commons.apache.org/logging/	Apache v1.1
Copyright © 1999-2001 The Apache Software Foundation.		
Crypto++ (AES/Rijndael and SHA-256)	http://www.cryptopp.com/	Public Domain
Copyright © 1995-2009 Wei Dai and contributors.		
Fast MD5	http://www.twmacinta.com/myjava/fast_md5.php	LGPL v2.1
Copyright © 2002-2005 Timothy W Macinta.		
HQP	http://hqp.sourceforge.net/	LGPL v2
Copyright © 1994-2002 Ruediger Franke.		
Jakarta Regexp	http://jakarta.apache.org/regexp/	Apache v1.1
Copyright © 1999-2002 The Apache Software Foundation.		
JavaHelp	http://javahelp.java.net/	GPL v2 with classpath exception
Copyright © 2011, Oracle and/or its affiliates.		
JXButtonPanel	http://swinghelper.dev.java.net/	LGPL v2.1 (or later)
Copyright © 2011, Oracle and/or its affiliates.		
LAPACK	http://www.netlib.org/lapack/	BSD Style
libodbc++	http://libodbcxx.sourceforge.net/	LGPL v2

Table of Contents

List of Figures

List of Tables

Chapter 1. Overview

Foreign Objects are external software components that provide certain computational services to gPROMS Models. These include physical property packages, external unit operation modules, or even complete computational fluid dynamics (CFD) software packages. This chapter explains how to make use of Foreign Objects that are already interfaced to gPROMS.

The Foreign Process Interface allows gPROMS simulations to interact with external software such as distributed control systems and operator training packages, exchanging data and other information. The interaction takes the form of a special set of elementary actions within the gPROMS Task language. The Foreign Process Interface is particularly useful for embedding gPROMS simulations within larger software systems.

Microsoft Excel ™Foreign Object and Foreign Process Interfaces are designed to allow gPROMS Activities to be configured using data provided in Microsoft Excel spreadsheets or to interact dynamically with calculations performed in Microsoft Excel.

Chapter 2. Using Foreign Objects

Introduction

What is a Foreign Object?

A Foreign Object is an external piece of software which provides a gPROMS simulation or optimisation run with computational services during run time. Typical examples include:

- A package that provides various routines for calculating the values of one or more thermophysical properties for given values of intensive and/or extensive variables in the gPROMS model.
- A spreadsheet that calculates the capital and operating cost for given values of equipment size and operating parameters in the gPROMS model.
- A set of routines for calculating mass transfer coefficients and other fluid transport properties for given values of intensive variables in the gPROMS model.

Why use a Foreign Object?

In principle, most types of mathematical relations can be expressed directly as Equations in gPROMS Models. However, there may still be good reasons why one would want to use an external ("foreign") software package to express relationships between some of the variables in a Model:

- The relationship cannot be expressed conveniently in closed algebraic form without the introduction of many intermediate quantities.
- The relationship involves many data parameters that would have to be extracted first and then inserted manually in the gPROMS Model.
- The software that carries out the required calculation already exists and is tested; it would be wasteful to have to reproduce its functionality in gPROMS.

A good example for all of the above reasons is provided by physical property calculations. Consider, for instance, the relationship between the specific enthalpy of a mixture and its temperature, pressure and composition implied by a cubic equation of state (e.g. the well-known Soave-Redlich-Kwong (SRK) equation). Expressing this relationship directly in the gPROMS language would involve the introduction of a number of auxiliary variables; this not only increases the size of the problem being solved but also sometimes makes its convergence more difficult: since many of these new variables have no direct physical meaning, it is difficult to obtain good initial guesses for them. Moreover, the equation of state involves critical point parameters for the various components in the mixture as well as binary interaction coefficients; these will have to be extracted from appropriate databanks and inserted in the gPROMS input file.

And yet all this effort is rather unnecessary if a well-documented and tested physical property package is already available. A better solution in this case would be to link the physical property package as a Foreign Object with gPROMS and make use of its services during the simulation.

What's in a name?

By this stage, you may well be wondering about the meaning and origins of the term "Foreign Object".

Here the term "Foreign" simply indicates that this software is external to the gPROMS system and its input.

The term "Object" is a software engineering one: it refers to the fact that this software is an independent block (a "component"), perhaps with its own private storage and internal workings; but the only way one can access any information relating to it is by making use of a well-defined interface that it provides to the outside world.

Some rules and terminology

Before going any further, let's introduce some of the rules that govern the use and operation of Foreign Objects in gPROMS and also some of the terminology that we will be using throughout this chapter.

- Each gPROMS simulation or optimisation run may interact with any number of Foreign Objects.
- Each Foreign Object provides a set of *methods.* These are simply the calculation routines that are accessible to the outside world. So, for instance, a typical physical property Foreign Object would provide methods for calculating vapour and liquid phase densities, enthalpies, fugacities and so on. And a Foreign Object for capital costing estimation could provide methods for calculating capital costs of pressure vessels, packed and tray distillation columns *etc.*
- A Foreign Object method in gPROMS calculates *one* quantity (the "output") for given values of one or more other quantities (the "inputs"). Consider, for instance, a typical method for calculating the specific enthalpy of a liquid-phase mixture provided by a physical property Foreign Object. This method would have the specific enthalpy as its unique output and the temperature, pressure and component mole fractions in the mixture as its three inputs.
- The output of a method can be either a scalar or a vector-and so can each one of its inputs. Going back to our physical property example, a method providing liquid-phase fugacities has a vector-valued output. And of course, the component mole fractions are a vector-valued input.
- The method may also provide partial derivatives of its output with respect to all or some of its inputs. Having access to such derivative information usually enhances both the efficiency and the robustness of the numerical computations carried out by gPROMS.

Classes and instances

As we mentioned at the start of the previous section on rules and terminology, each gPROMS run may make use of any number of Foreign Objects. So, by way of an example, consider a simulation run making use of:

- a Foreign Object calculating physical properties in the high-pressure, low-temperature (cryogenic) region;
- a Foreign Object calculating physical properties in the low-pressure, ambient-temperature region;
- a spreadsheet object that calculates capital and operating costs; and
- another spreadsheet that does accounting calculations of tax liabilities and asset depreciation.

Now, the two physical property Foreign Objects mentioned above may, in fact, involve the *same* physical property software, say a new package called *SUPERPRO*; thus, they make available exactly the same types of physical property calculations (density, enthalpy, fugacities - all of the "methods"). However, they do differ in the data and correlations that they use internally for each one of these calculations-perhaps, even in the set of components that appear in the mixtures that they are supposed to handle.

In such a situation, we would say that these two Foreign Objects are different *instances* (one of which is, say, called "HPCryoProps" and the other "LPAmbientProps") of the same class, namely *SUPERPRO*.

Similarly, the two spreadsheet Foreign Objects mentioned above may actually employ the same spreadsheet software (e.g. Microsoft Excel™; but, of course, they use different spreadsheet files (e.g. one called Costing.xls and another one called Tax.xls respectively). Again, in this case, we would have two instances, say called Costing and Tax, both belonging to the same class *ExcelFO*).

What do I need to read next?

If you intend to use Foreign Objects that are *already* interfaced to gPROMS, all you need to read is the next section. This tells you how to reference Foreign Objects inside your gPROMS input files.

If, on the other hand, you intend to interface your own Foreign Objects to gPROMS for use by yourself and/or other people, you will first need to read the next section on using Foreign Objects and then read the chapter entitled "Developing New Foreign Objects" in the "gPROMS System Programmer Guide".

As you may have already concluded from the examples we used above, one of the most common uses of Foreign Objects is as a means of interfacing physical property packages to gPROMS. Indeed, this is so common that gPROMS provides a simplified mechanism for using and implementing such interfaces. If all you want to do is learn how to use the physical property interface, then you should read chapters on using Physical Properties for simple and complex materials. The implementation of *new* physical properties interfaces is discussed in the chapter entitled "Developing New Physical Properties Interfaces" in the "gPROMS System Programmer Guide".

Using Foreign Objects in gPROMS entities

A description of how to make use of Foreign Objects in different gPROMS entities follows. It is assumed that these objects are already fully implemented, tested and interfaced to gPROMS.

Using Foreign Objects in gPROMS Models

As we have already explained, a Foreign Object basically provides a means of calculating one or more quantities as function(s) of the gPROMS variables. As such, the natural place for these objects to appear is in the gPROMS Models.

The usage of Foreign Objects in Models is governed by a set of simple rules:

A. Each distinct Foreign Object used by a Model is declared as a Parameter of type FOREIGN_OBJECT[1]

The latter keyword is normally followed by the *class* of the Foreign Object (see also: Classes and Instances). This class simply identifies the external software (e.g. physical software package, CFD code, capital costing tool) that will be used to implement this instance of the Foreign Object. For example, a typical declaration would be:

```
# MODEL Flash

    PARAMETER
      PPP AS FOREIGN_OBJECT "ThermoPack"

    ...
```

This simply states that:

- Model Flash will make use of a Foreign Object.
- Whenever necessary within this Model (see below), the Foreign Object will be referred to as PPP.
- PPP is implemented by a piece of external software (in this case, a physical property package) called ThermoPack. Thus, PPP is an "instance" of ThermoPack.

One important thing to note is that the class of the Foreign Object must be enclosed in quotes ("ThermoPack"). This is because it is neither a gPROMS keyword nor an identifier that has previously been defined in the gPROMS input file.

B. The methods of a Foreign Object are referred to as:

```
ForeignObjectName.MethodName (InputList)
```

where the InputList is a list of method inputs. For a method without any inputs, the correct syntax is:

```
ForeignObjectName.MethodName
```

[1]This is a new type of Parameter complementing the existing types INTEGER, REAL and LOGICAL.

C. Each input of a method is a scalar or vector-valued variable or expression.

D. Each method returns a single scalar or vector-valued quantity. The latter may be of type integer, logical or real. A method may be used *anywhere* in the MODEL where an expression of the corresponding dimensionality and type is allowed. For instance:

- A real-valued method may be used in an equation or in an expression Setting the value of a real Parameter.
- An integer-valued method may be used to Set the value of an integer Parameter.
- A logical-valued method may be used within the logical condition in an If equation.

An example of a gPROMS Model making use of a Foreign Object is shown in the example below:

```
1 # MODEL Flash

 2     PARAMETER
 3       PPP AS FOREIGN_OBJECT "ThermoPack"
 4       NoComp AS INTEGER

 5     VARIABLE
 6       F, L, V AS MolarRate
 7       Hf AS MolarEnergy
 8       Q AS EnergyRate
 9       T AS Temperature
10       P AS Pressure
11       x, y, z AS ARRAY(NoComp) OF MoleFraction

12     SET
13       NoComp := PPP.NumberOfComponents ;

14     EQUATION
15       F*z = L*x + V*y ;
16       F*Hf = L*PPP.LiquidEnthalpy(T,P,x) +
                                   V*PPP.VapourEnthalpy(T,P,y) + Q ;
17       x*PPP.LiquidFugacityCoeff(T,P,x) =
                                  y*PPP.VapourFugacityCoeff(T,P,y) ;
18       SIGMA(x) = SIGMA(y) = 1 ;
19       IF PPP.StableLiquid(T,P,x) THEN
20         ...
21       ELSE
22         ...
23       END
```

There are several features of this Model that are worth noting:

- As we have already seen, line 3 specifies that the Model Flash makes use of a Foreign Object of type ThermoPack, corresponding to a software package that is external to gPROMS. For the purposes of defining this Model, this Foreign Object will be called PPP.
- This Foreign Object provides several methods. These return quantities of various types that can be used anywhere in the Model. For example:
  - At line 13, we make use of a method called NumberOfComponents which simply returns an integer corresponding to the number of components in the mixture being considered. We use this to Set the number of components, a Parameter NoComp, in this Model. Note that method NumberOfComponents does not have any inputs.
  - At line 16, we make use of two other methods provided by this Foreign Object, namely LiquidEnthalpy and VapourEnthalpy, which compute liquid and vapour phase specific enthalpies respectively. Each of these

returns a real quantity computed by the Foreign Object from the inputs T,P,x and T,P,y respectively. We note that, in each case, the first two inputs are scalar quantities but the third one is a vector.

- At line 17, we make use of two more methods, namely LiquidFugacityCoeff and VapourFugacityCoeff respectively. These compute and return *vectors* of real quantities, *i.e.* liquid and vapour phase fugacity coefficients.
- At line 19, we make use of a method called StableLiquid that returns a logical quantity indicating whether or not the liquid phase is stable under the prevailing temperature, pressure and composition.

There are two further points that are worth making:

- Method inputs can be expressions rather than simple variables. For instance, if we wanted to evaluate a liquid-phase viscosity at the mean of the inlet and exit conditions of the above flash, we could achieve this via the following method invocation:

  ```
  PPP.LiquidViscosity((Tin+T)/2, (Pin+P)/2, (z+x)/2)
  ```

- No direct reference can be made to an individual element (or slice) of a vector-valued method. For instance, it is not possible to directly access the i^{th} element of the vector of liquid fugacity coefficients returned by method LiquidFugacityCoeff (see line 17 in the example above). If such access is desired, an equation defining a vector-valued variable as being equal to the method result must first be introduced, e.g.:

  ```
  LFC = PPP.LiquidFugacityCoeff(T.P,x) ;
  ```

  Other equations can then refer to individual elements of the variable LFC as desired.

Foreign Object values and their specification

As we have seen, each Foreign Object is an instance of a class which determines the actual external software that will be used to compute the methods that the Foreign Object is supposed to provide. However, in most cases, just specifying the class of the Foreign Object is not sufficient to determine completely its behaviour.

Consider, for instance, the Foreign Object PPP used in the example below. We already know that the methods (e.g. LiquidFugacityCoeff) of this Foreign Object will be provided by an external physical property package called ThermoPack. However, for these methods to be fully defined,we also need to know the set of components that are involved in the mixture under consideration and also the particular type of thermophysical model (e.g. the equation of state or activity coefficient model) that will be used to compute these properties. All of this information is associated with the specific Foreign Object instance PPP. Indeed, two different instances of the same Foreign Object class within the same gPROMS simulation may involve different component sets or use different thermophysical property calculation options.

As described in: Using Foreign Objects in gPROMS Models, the Foreign Objects used in a Model are declared as Parameters. Therefore, just like any other Parameter, each Foreign Object must eventually be given a "value" before any instance of the Model can be used in a simulation or optimisation run.

More specifically, the value of the Foreign Object is a string of characters (enclosed in double quotes) that identifies this particular instance of a Foreign Object. In the example described above, this string could be a pathname for a data file that contains the information on the component set and thermophysical property calculation options. The ThermoPack software would read this file to determine precisely what mixture it is supposed to handle and how. Different instances of ThermoPack would be described by different such files.

```
1 # MODEL Flash

 2     PARAMETER
 3       PPP AS FOREIGN_OBJECT "ThermoPack"
 4       NoComp AS INTEGER

 5     VARIABLE
```

```
 6       F, L, V AS MolarRate
 7       Hf AS MolarEnergy
 8       Q AS EnergyRate
 9       T AS Temperature
10       P AS Pressure
11       x, y, z AS ARRAY(NoComp) OF MoleFraction

12     SET
13       NoComp := PPP.NumberOfComponents ;

14     EQUATION
15       F*z = L*x + V*y ;
16       F*Hf = L*PPP.LiquidEnthalpy(T,P,x) +
                                    V*PPP.VapourEnthalpy(T,P,y) + Q ;
17       x*PPP.LiquidFugacityCoeff(T,P,x) =
                                   y*PPP.VapourFugacityCoeff(T,P,y) ;
18       SIGMA(x) = SIGMA(y) = 1 ;
19       IF PPP.StableLiquid(T,P,x) THEN
20         ...
21       ELSE
22         ...
23       END
```

Explicit specification of a Foreign Object value

Like any other gPROMS Parameter, the value of a Foreign Object can be Set in one of three ways:

A. Within the Model in which the Foreign Object is declared.

   For example, we could insert the following line after line 12 of the Flash Model shown in the example in : Foreign Object values and their specification

   ```
   PPP := "Aromatics.tpk" ;
   ```

   This states that the behaviour of Foreign Object PPP is defined by a ThermoPack input file (.tpk) called Aromatics.tpk. The input file must be imported into ModelBuilder as a Miscellaneous File (or linked) - see the ModelDeveloper User Guide. Alternatively the full name for any file located outside ModelBuilder can be used, e.g.

   ```
   C:\Temp\Aromatics.tpk
   ```

   The obvious disadvantage of the above way of specifying the Foreign Object value is that *all* instances of Model Flash will be restricted to using the same set of components and thermophysical property calculation options. Thus, the generality and re-usability of this Model is severely limited!

B. Within a higher-level Model containing instances of the Model within which the Foreign Object was declared.

   Consider, for instance, the Model TwoTankSystem below that illustrates Setting Foreign Object values within composite Models.

   ```
   1 # MODELTwoTankSystem

    2   PARAMETER
    3     LPPP AS FOREIGN_OBJECT "ThermoPack"

    4   VARIABLE
    5     AverageMolWt AS MolecularWeight

    6   UNIT
    7     HighPTank AS Flash
   ```

```
 8     LowPTank  AS Flash

 9   SET
10     HighPTank.PPP := "HPThermo.tpk" ;
11     LowPTank.PPP  := LPPP ;

12   EQUATION
13     AverageMolWt * (HighPTank.L + LowPTank.L) =
14       (HighPTank.L*HighPTank.PPP.MeanMW(HighPTank.x) +
15       LowPTank.L*LowPTank.PPP.MeanMW(LowPTank.x))/2 ;
```

It contains two instances,HighPTank and LowPTank respectively, of the Model Flash - the example used earlier in the chapter. These two flashes operate at significantly different pressures, and consequently make use of different thermophysical property calculation options. At line 10, the Foreign Object PPP of the first of these two flashes is identified with a ThermoPack input file HPThermo.tpk. On the other hand, line 11 sets the corresponding Foreign Object of the second flash to be equal to a Parameter LPPP declared within TwoTankSystem (see line 3). Obviously, LPPP will have to be given a value at an even higher-level Model or in a Process (see below).

As is standard in gPROMS, a composite Model can access any entity declared within instances of lower-level Models that it contains. This rule also holds for Foreign Object entities. For example, in lines 13-15 above, the PPP Foreign Objects within HighPTank and LowPTank are used in order to compute the average molecular weight of the liquid streams leaving the two units.

C. Within the Process section defining the simulation or optimisation run.

Setting a Foreign Object value within a Process entity is illustrated below. Lines 2-3 declare an instance (called Plant) of the composite Model defined in above. The value of LPPP occurring in that Model is then Set in lines 4-5.

```
1 # PROCESS PlantSimulation

2   UNIT
3     Plant AS TwoTankSystem

4   SET
5     Plant.LPPP := "LPThermo.tpk" ;

6   ...
```

Implicit specification of a Foreign Object value

Like other gPROMS Parameters, the values of Foreign Objects may be specified implicitly via the gPROMS parameter propagation mechanism.

Parameter propagation is invoked automatically by gPROMS at the start of the execution of a Process entity if the value of any Foreign Object instance within the entire problem has not been specified explicitly. In such cases, gPROMS will try to determine the missing value by searching for a Foreign Object that:

- is declared within a higher-level Model containing the instance of the Model which has the unspecified Foreign Object, **and**
- has exactly the same name as the unspecified Foreign Object, **and**
- belongs to the same class as the unspecified Foreign Object, **and**
- has already been given a value either explicitly or implicitly.

If a Foreign Object fulfilling *all* of the above criteria is found, its value is also given to the unspecified Foreign Object. If no such Foreign Object is found, the algorithm is applied recursively, searching increasingly higher-level Models and ultimately the Process itself. If, by the end of this procedure, the value of the unspecified Foreign Object remains undetermined, an error occurs and the execution of the Process is aborted.

The main practical implication of the parameter propagation mechanism is that, provided the model developer adopts a standard name (say PPP) for all Foreign Object instances of a certain class (e.g. ThermoPack) in all Models that he or she develops, then it is sufficient to specify the value of this Foreign Object *once only* (usually in the Process entity) for it to be adopted automatically by the entire problem.

Alternatively, if a plant has two distinct sections (e.g. a high temperature and a low temperature one) which require the use of two different Foreign Objects, this can again be specified conveniently at the highest appropriate level, as illustrated below - Foreign Object specification via parameter propagation. Here, it is assumed that the Models HighTemperatureSection and LowTemperatureSection referred to in lines 3-4 each contain a Foreign Object called PPP and so do all of their sub-models. In this case, simply Setting appropriate values for the two highest-level Foreign Objects in the Process entity (see lines 5-7) achieves the desired specification for the entire problem.

```
1 # MODEL Plant

2   UNIT
3     HTS AS HighTemperatureSection
4     LTS AS LowTemperatureSection

5   SET
6     HTS.PPP := "HighTThermo.tpk" ;
7     LTS.PPP := "LowTThermo.tpk" ;

8   EQUATION
9     ...
```

Consistency checking and validation

Ensuring correct usage of Foreign Objects

To make correct use of a Foreign Object in your Models, you need the following information:

A. The name of the Foreign Object class.

B. The names of the methods provided by this Foreign Object.

C. The following information for each method:

i. The type, length, physical dimensions and units of measurement of the method output.

ii. The number of the method inputs.

iii. The length, physical dimensions and units of measurement of each of the method inputs.

It is worth noting the following in conjunction with points (C-i) and (C-iii) above:

- The *type* of a quantity is real, integer or logical.
- The *length* of a quantity is 1 for a scalar, and *n* for a vector (array) of length *n*.
- The *physical dimensions* are defined only for *real* quantities. They determine the physical type of the quantity in terms of the nine fundamental physical dimensions as well as a tenth dimension, "money" (see the table below). For instance, a molar density has dimensions:

  [Amount of Substance]^{1}[Length]$^{-3}$

  A unit material cost might be expressed in terms of:

  [Money]^{1}[Mass]$^{-1}$

- The *units of measurement* are expressed in relation to the SI set of units (see last column of the table below) in the following manner:

  $$\left(\begin{array}{c}\text{Value of}\\ \text{Quantity in}\\ \text{SI Units}\end{array}\right) = \text{Offset} + \text{Multiplier} \times \left(\begin{array}{c}\text{Value of}\\ \text{Quantity Within}\\ \text{Foreign Object}\end{array}\right)$$

  where the Offset and the Multiplier are two real-valued quantities. For instance, if the Foreign Object measures temperatures in degrees Fahrenheit, then the Offset is 255.37 and the Multiplier is 0.55556.

You should be able to obtain all of the above information from the documentation accompanying the particular Foreign Object that you wish to use.

Table 2.1. Physical dimensions for the inputs and output of a method.

Fundamental Dimension	Description	SI Units
1	Length	metre
2	Mass	kilogram
3	Time	second
4	Electric Current	Ampere
5	Temperature	Kelvin
6	Amount of Substance	mole
7	Luminous Intensity	candela
8	Plane Angle	radian
9	Solid Angle	steradian
10	Money	US dollar

Consistency checking

gPROMS checks the consistency of your Foreign Object usage at the start of the execution of a Process. To achieve this, it goes though a number of steps:

1. It attempts to locate the external software identified by the Foreign Object class name(s) by searching first in a sub-directory called "fo" in the export directory;[2]

   if it does not find it there, it searches in a directory specified by the operating system "environment variable" GPROMSFODIR if the latter has been given a value. Finally, it searches in a sub-directory called "fo" of the gPROMS system installation directory.

   If the external software cannot be located or if it can be located but communication with gPROMS cannot be established for whatever reason, execution of the Process will abort with a message of the form:

   ```
   Cannot open the Foreign Object dynamic shared library:<FOClass.so>
   Foreign Object initialisation failed: <FOClass>
   ```

   where <FOClass> is the name of the Foreign Object class whose software implementation cannot be found.

2. It establishes a link with the external software, passing to it the value (see also: Foreign Object values) of each Foreign Object instance of this class and asking it to create such an instance.

   If the instance value is illegal (e.g. because it contains illegal characters or it refers to a file that cannot be located by the external software), then execution of the PROCESS will abort with the message:

   ```
   Cannot create Foreign Object instance:<FOClass>::<FOInstance>
   ```

[2]The foreign object must be imported into Modelbuilder or linked, with the export directory set to fo. See the ModelDeveloper User Guide.

```
Foreign Object initialisation failed: <FOClass>::<FOInstance>
```

where <FOInstance> is the name of the Foreign Object instance of class <FOClass> that is causing the problem.

3. It compares the usage of each Foreign Object instance in the problem against information that it obtains by issuing direct requests to that instance. For example, it will seek to verify that any methods of this Foreign Object that are referred to in Models are really supported by the external software, and that the actual type and length of their outputs are consistent with the manner in which they are used in these Models. It also checks the number and length of the inputs of each such method.

Warning

gPROMS does *not* currently check the consistency of the fundamental physical dimensions or the units of measurement, nor does it carry out automatic conversion of units of measurement. It is your responsibility to ensure the correct usage of Foreign Objects in this respect.

Chapter 3. Using Foreign Processes

Introduction

The Foreign Process Interface (FPI) provides a general mechanism for the exchange of information between executing gPROMS simulations and external software.

This communication takes place at discrete time points throughout the duration of the simulation. The user is entirely free to determine the frequency and content of the exchanges, which may include:

- time synchronisation signals;
- values of variables and flags;
- information on the mathematical model used for the simulation and its current state.

Applications of the FPI

Typical applications of the FPI include:

- the use of gPROMS simulations to study and validate control algorithms implemented in an external real-time control system;
- the construction of bespoke user-interfaces to gPROMS simulations offering a well-defined and limited scope for modifications on behalf of the end-user;
- the incorporation of gPROMS within operator training systems;
- the interfacing of gPROMS with model-based control system design packages.

What *is* the FPI?

The FPI comprises two main components:

- *A set of elementary communication tasks.*

  By inserting instances of these tasks in Schedules within Task and Process entities, the user determines the timing and content of any communication that will occur when the Schedule is executed.

- *A communication protocol between gPROMS and the external software.*

  The execution of an elementary communication task in a Schedule automatically causes gPROMS to invoke one of a set of procedures provided by the user.

  The FPI communication protocol specifies

  - what procedures must be provided;
  - their names;
  - the precise form of their argument lists.

  The FPI communication protocol does *not* specify the content or detailed *behaviour* of these procedures. They can be as complex or as simple as desired-in fact, some of them may be completely empty!

  Typically, the communication procedures will be developed *once* for a given hardware/software setup.

Chapter outline

The rest of this chapter is structured as follows:

- The section on elementary communication tasks describes the elementary communications tasks that comprise an FPI and the effects they have on the simulation.
- The section on using the FPI explains how to ensure that the correct FPI implementation is used by your simulation.
- Finally, the section on FPI performance issues discusses some important issues pertaining to the performance of simulations that make use of the FPI.

This chapter does *not* discuss how to develop new FPI implementations. This topic is covered in detail in the chapter entitled "Developing New FPI Implementations" in the "gPROMS System Programmer Guide".

The elementary communication tasks

Dynamic simulations in gPROMS are defined in Process entities which can be executed to carry out the simulation. In general, each Process involves a Schedule of actions (defined by Task entities) organised in Sequence or in Parallel. There is also the possibility of conditional (If) and iterative (While) task execution.

A Task entity may be defined in terms of lower level tasks which, in turn, may involve even lower level tasks, and so on. We therefore have a hierarchy of tasks of arbitrary depth. At the bottom level of this hierarchy, we have a number of *elementary* tasks, such as Reset, Replace and Continue (see the chapters entitled Simple Operating Procedures and Complex Operating Procedures in the gPROMS Model Developer Guide).

The FPI introduces five new elementary tasks. Just like the other elementary tasks, these can be inserted anywhere in a Schedule. Their function is to exchange information between the executing gPROMS Process and some external software. The latter, as far as gPROMS is concerned, is just a "foreign" Process (i.e. one that is *not* written in the gPROMS language). From now on, we will use this term to refer to the external software.

The five elementary communication tasks are summarised below:

PAUSE	Hold the gPROMS simulation until receiving a signal from the Foreign Process.
GET	Receive the values of a subset of the gPROMS variables from the Foreign Process.
SEND	Transmit the values of a subset of the gPROMS variables to the Foreign Process.
SENDMATHINFO	Transmit information on the mathematical model used for the simulation, and its current state, to the Foreign Process.
LINEARISE	Create the minimal state-space representation of a non-linear, dynamic model

All elementary communication tasks are executed in a *synchronous* fashion. That is, the simulation is suspended completely (i.e. the simulation clock is stopped) while gPROMS is waiting for the response of the foreign process.

In the rest of this section, we consider each of these elementary tasks in more detail.

The Pause elementary communication task

The Pause elementary task simply holds the gPROMS simulation until a signal is received from the Foreign Process.

General syntax

The syntax of the Pause elementary task is:

```
PAUSE SIGNALID "SigName" STATUS StatVar
```

where

SigName	is a string of characters that will be passed to the Foreign Process by gPROMS on executing the Pause task. This may be used by the Foreign Process for identifying the actual Pause request being executed, which may be useful if there are several Pause tasks at different places in the Schedule. The Signalid "*SigName*" specification is optional. An empty string will be passed to the Foreign Process by default.
StatVar	is a logical gPROMS variable[a]that will receive the outcome of the Pause request from the Foreign Process. Generally, a value of True will indicate success (or permission to proceed with the simulation), while a value of False may signal failure (or a request to abort the simulation). The Status *StatVar* specification is also optional.

[a]Logical variables are normally defined within the Variable section of Task entities.

A typical use of a Pause task is right at the beginning of a Task Schedule:

```
# TASK Example

  VARIABLE
    OKtoContinue   AS    LOGICAL

  SCHEDULE
    SEQUENCE
      PAUSE SIGNALID "ExampleStartRequest" STATUS OKtoContinue
      IF  OKtoContinue  THEN
        SEQUENCE
         { Do rest of task }
        END
      ELSE
        SEQUENCE
          MESSAGE "TASK Example aborted !"
          STOP
        END
      END
    END
```

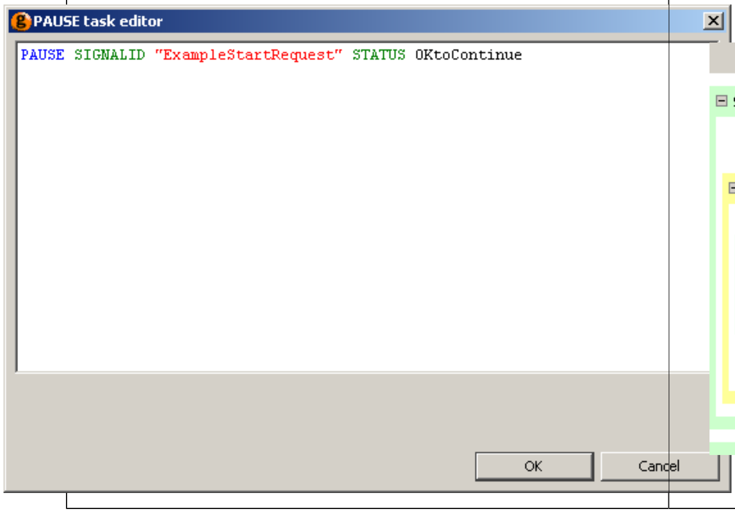
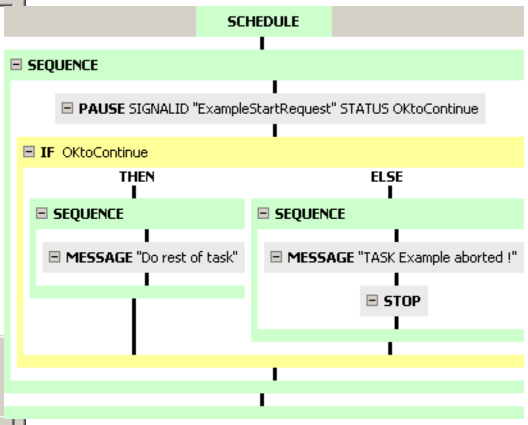
In this case, the first action of Task Example is to Pause, sending the string of characters "ExampleStartRequest" to the Foreign Process. The simulation clock stops at this point waiting for the Foreign Process to respond. The execution of the rest of the Task depends on the value of the Status variable returned by the Foreign Process.

Using Pause in the Schedule tab of a Task or Process

As with all other elementary Tasks and user-defined Tasks, the Pause Task can be dragged into a Schedule using the Task palette or by right clicking on a hot spot in the Schedule and selecting Pause from the Add Foreign Process task from the context menu.

When this is done, a dialog will appear so that the Pause Task can be configured. Type in the gPROMS language for the Task and press the OK button. The dialog for the example presented before is shown below, along with the Schedule.

Table 3.1. Example of the Pause Task

Dialog	In Schedule
 PAUSE task editor: `PAUSE SIGNALID "ExampleStartRequest" STATUS OKtoContinue`	 SCHEDULE SEQUENCE PAUSE SIGNALID "ExampleStartRequest" STATUS OKtoContinue IF OKtoContinue THEN ELSE SEQUENCE SEQUENCE MESSAGE "Do rest of task" MESSAGE "TASK Example aborted !" STOP

Side effects of Pause

Pause tasks have no side effects on the simulation results.

The Get elementary communication task

The Get elementary task receives values of one or more variables from the Foreign Process.

General syntax

The syntax of the Get elementary task is:

```
GET
  SIGNALID "SigName"
  STATUS StatVar
    gPROMSVariable ;
    gPROMSVariable ;
    gPROMSVariable ;
    ...
    gPROMSVariable := "ForeignVariableID" ;
    gPROMSVariable := "ForeignVariableID" ;
    gPROMSVariable := "ForeignVariableID" ;
    ...
END
```

where

SigName	is a string of characters that will be passed to the Foreign Process by gPROMS on executing the Get task. This may be used by the Foreign Process for identifying the actual Get request being executed, which may be useful if there are several Get tasks at different places in the Schedule. In some applications in which the gPROMS Process is expected to receive information from more than one

	foreign source, the *SigName* may also be used to identify the particular source of the data requested. It is the responsibility of the FPI implementation to request the data from the correct source (e.g., in an operator training context, from a trainee's console rather than from the trainer's console) and return them to gPROMS. The Signalid "*SigName*" specification is optional. An empty string will be passed to the Foreign Process by default.
StatVar	is a logical gPROMS variable that will receive the outcome of the Get request from the Foreign Process. Generally, a value of True will indicate success (or permission to proceed with the simulation), while a value of False may signal failure (or a request to abort the simulation). The Status *StatVar* specification is optional.
gPROMSVariable	is either a variable occurring in a gPROMS Model entity used for the simulation (specified through its complete pathname), or a local variable in a Task entity (of type Real, Integer, or Logical).
ForeignVariableID	is the name by which this quantity is known to the Foreign Process (e.g. a "tag name" in a real-time database). This will be used by the Foreign Process to identify the requested information so that it can return it to gPROMS. If the *ForeignVariableID* is omitted, the complete path name of *gPROMSVariable* will be used instead.

Getting array and distribution slices

In most applications, Get is applied to individual variables or array elements. However, in some cases, it may be desirable to Get values of an entire array or a slice of an array. The syntax for achieving this is of the form:

```
GET
  SIGNALID "SigName"
  STATUS StatVar
  gPROMSArrayVariable := "ForeignVariableID"(1:N) ;
  gPROMSArrayVariable(N1:N2) := "ForeignVariableID"(1:N2-N1+1) ;
  gPROMSArrayVariable ;
  gPROMSArrayVariable(N1:N2) ;
  ...
END
```

The main point to note is that if a foreign variable ID is specified to receive an array of gPROMS variable values, then we must declare this *ForeignVariableID* as an *array* of the same length. Thus, in the first case above, to get an *entire* array of length *N*, we treat the *ForeignVariableID* also as an array of length *N and* show this explicitly by adding the (1:N) "slice" notation to it.

Similarly, in the second case, to get the values of a slice of the array between indices *N1* and *N2* inclusive (assuming $N1 \leq N2$), we treat the *ForeignVariableID* as an array of the same length(i.e. comprising *N2 - N1 + 1* elements).

If entire multi-dimensional arrays or slices thereof are to be received, the *ForeignVariableID* must be treated as arrays of the same number of dimensions *and* this must be shown explicitly. For instance:

```
GET
  SIGNALID "SigName"
  STATUS StatVar
  gPROMSArrayVariable(N1:N2, N3:N4) := "ForeignVariableID" (1:N2-N1+1, 1:N4-N3+1) ;
  gPROMSArrayVariable(N1:N2, N3:N4) ;
  ...
END
```

Similar syntax and rules apply for Getting variables that are distributed over one or more *continuous* domains. However, in such cases, it must be borne in mind that Get can only obtain a *discrete* representation of the variable:

```
GET
  SIGNALID "SigName"
  STATUS StatVar
  gPROMSDistributedVariable := "ForeignVariableID"(1:N) ;
  gPROMSDistributedVariable ;
  ...
END
```

where *N* is the number of elements in the discretisation of the distributed variable. This will depend on the discretisation method specified for the corresponding distribution domain, which is normally of the form:

```
SET DistributionDomain := [ Method, Order, NumberOfElements ] ;
```

In particular,

- for finite-difference based methods (i.e. for *Method* being BFDM, CFDM, or FFDM), use

  $$N = NumberOfElements + 1$$

  Here the values to be received correspond to the values of the distributed variable at *N* uniformly distributed points (including the domain boundaries).

- for orthogonal-collocation based methods (i.e., for *Method* being OCFEM), use

  $$N = NumberOfElements \times Order + 1$$

  Here the values to be received correspond to the values of the distributed variable at the (*NumberOfElements + 1*) finite element boundaries, and at the (*Order - 1*) collocation points within each finite element.

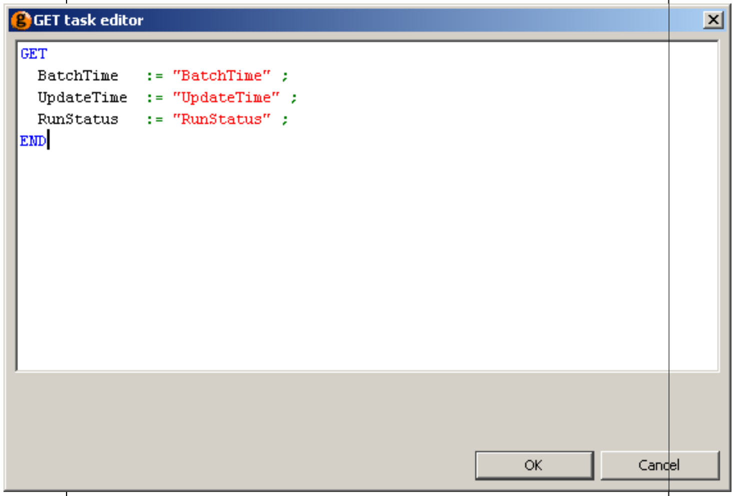
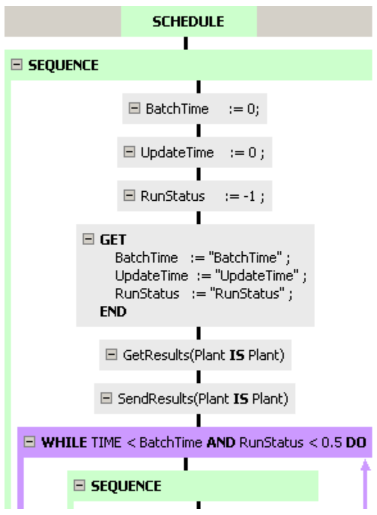
Getting *slices* of distributed variables is *not* recommended as it requires a detailed understanding of the internal representation and handling of variable distributions by gPROMS.

Using Get in the Schedule tab of a Task or Process

As with all other elementary Tasks and user-defined Tasks, the Get Task can be dragged into a Schedule using the Task palette or by right clicking on a hot spot in the Schedule and selecting Get from the Add Foreign Process task from the context menu.

When this is done, a dialog will appear so that the Get Task can be configured. Type in the gPROMS language for the Task and press the OK button. The dialog for the PML Batch Plant example is shown below, along with the Schedule.

Table 3.2. Example of the Get Task

Dialog	In Schedule
	

```
GET
  BatchTime   := "BatchTime" ;
  UpdateTime  := "UpdateTime" ;
  RunStatus   := "RunStatus" ;
END
```

```
SCHEDULE
SEQUENCE
  BatchTime   := 0;
  UpdateTime  := 0 ;
  RunStatus   := -1 ;
  GET
    BatchTime  := "BatchTime" ;
    UpdateTime := "UpdateTime" ;
    RunStatus  := "RunStatus" ;
  END
  GetResults(Plant IS Plant)
  SendResults(Plant IS Plant)
  WHILE TIME < BatchTime AND RunStatus < 0.5 DO
    SEQUENCE
```

Side effects of Get

A successful return (i.e. *StatVar* = TRUE) from the foreign process following a Get request provides the gPROMS simulation with a set of new values for the specified subset of its variables. These values overwrite the values that the variables had before the Get task. The actual effect of this overwriting on the subsequent simulation depends on the type of each such variable:

- If *gPROMSVariable* is a local Task variable, then the new value simply overwrites the old value without any other side effect. In effect, this is equivalent to:

  gPROMSVariable := new value ;

- If *gPROMSVariable* is an input variable[1]

  in the mathematical model, being currently Assigned to a numerical value or an explicit function of time, then it becomes Assigned to the new value obtained via Get In effect, this is equivalent to:

  ```
  RESET
    gPROMSVariable := new value ;
  END
  ```

- If *gPROMSVariable* is a differential variable in the mathematical model, then its current value is changed instantaneously to the new value obtained via Get. In effect, this is equivalent to:

  ```
  REINITIAL
    gPROMSVariable
  WITH
    gPROMSVariable = new value ;
  END
  ```

- If *gPROMSVariable* is an algebraic variable in the mathematical model, then its current value is momentarily overwritten by the new value.

If the Get task changes the values of any input or differential variables in the model, then these new values may be inconsistent with the values of all other model variables (i.e. they no longer satisfy the model equations). In such cases, gPROMS automatically undertakes a re-initialisation calculation which aims to restore this consistency while keeping all input and differential variables at their new values.

[1]The classification of gPROMS variables into input, differential and algebraic variables is discussed in detail in: information transmitted by Sendmathinfo.

An interesting consequence of the way this re-initialisation is done is that Getting an *algebraic* variable has absolutely no effect on the subsequent trajectory that the simulation follows: irrespective of what the value obtained by Get is, the variable will be restored to a value that is consistent with the values of the input and differential variables.

However, there *are* circumstances under which Getting one or more algebraic variables may be useful. In particular, it should be borne in mind that re-initialisation involves the solution of a non-linear set of equations which is carried out iteratively starting from the current point as the initial guess. A good initial guess for a critical algebraic variable (e.g. a temperature in a chemical reactor) received from the Foreign Process (e.g. a control system having access to real plant measurements) may facilitate the convergence of this iteration-even if it does not affect the converged solution found eventually.

The Send elementary communication task

The Send elementary task sends values of one or more variables to the Foreign Process.

General syntax

The syntax of the Send elementary task is:

```
SEND
  SIGNALID "SigName"
  STATUS StatVar
  gPROMSVariable ;
  "ForeignVariableID" := gPROMSVariable ;
  gPROMSVariable ;
  "ForeignVariableID" := gPROMSVariable ;
  gPROMSVariable ;
  "ForeignVariableID" := gPROMSVariable ;
  ...
END
```

where

SigName	is a string of characters that will be passed to the Foreign Process by gPROMS on executing the Send task. This may be used by the Foreign Process for identifying the actual Send request being executed, which may be useful if there are several Send tasks at different places in the Schedule. In some applications in which the gPROMS Process is expected to transmit information to more than one foreign destination, the *SigName* may also be used to identify the final destination of the data being sent. It is the responsibility of the FPI implementation to direct the data to the correct destination (e.g., in an operator training context, to a trainee's console rather than to the trainer's console). The Signalid "*SigName*" specification is optional and may be omitted. An empty string will be passed to the foreign process by default.
StatVar	is a logical gPROMS variable that will receive the outcome of the Send request from the Foreign Process. Generally, a value of True will indicate success (or permission to proceed with the simulation), while a

	value of False may signal failure (or a request to abort the simulation). The Status *StatVar* specification is optional.
ForeignVariableID	is the name by which this quantity is known to the Foreign Process (e.g. a "tag name" in a real-time database). This will be used by the Foreign Process to identify the requested information so that it can return it to gPROMS. If the *ForeignVariableID* is omitted, the complete path name of *gPROMSVariable* will be used instead.
gPROMSVariable	is either a variable occurring in a gPROMS Model entity used for the simulation (specified through its complete pathname), or a local variable in a Task entity (of type Real, Integer, or Logical).

Sending array and distribution slices

Sending entire arrays and distributions, or slices thereof to a foreign process is handled in an entirely analogous fashion to Getting them from the Foreign Process.

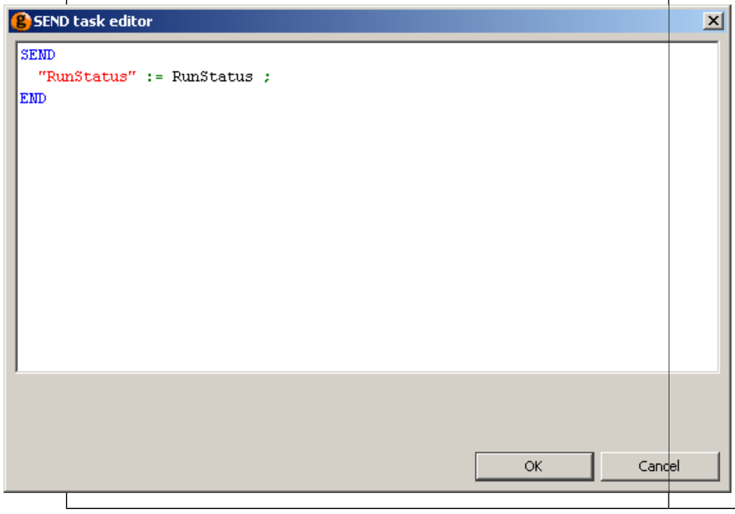
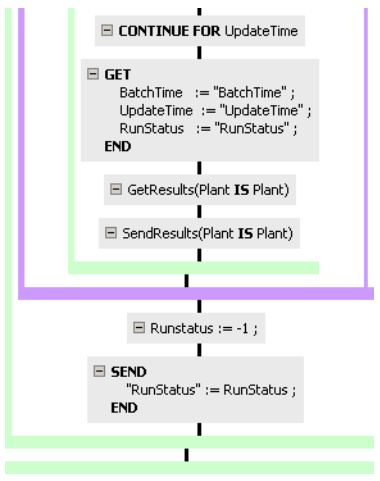
See also: Getting array and distribution slices for details on the relevant syntax and rules.

Using Send in the Schedule tab of a Task or Process

As with all other elementary Tasks and user-defined Tasks, the Send Task can be dragged into a Schedule using the Task palette or by right clicking on a hot spot in the Schedule and selecting Send from the Add Foreign Process task from the context menu.

When this is done, a dialog will appear so that the Send Task can be configured. Type in the gPROMS language for the Task and press the OK button. The dialog for the PML Batch Plant example is shown below, along with the Schedule.

Table 3.3. Example of the Send Task

Dialog	In Schedule
	

```
SEND
  "RunStatus" := RunStatus ;
END
```

```
CONTINUE FOR UpdateTime
GET
    BatchTime  := "BatchTime" ;
    UpdateTime := "UpdateTime" ;
    RunStatus  := "RunStatus" ;
END
GetResults(Plant IS Plant)
SendResults(Plant IS Plant)
Runstatus := -1 ;
SEND
    "RunStatus" := RunStatus ;
END
```

Side effects of Send

Send tasks have no side effects on the simulation results.

The Sendmathinfo elementary communication task

The Sendmathinfo elementary task transmits to the Foreign Process information on the mathematical model being used by the simulation, and its current state.

General syntax

The syntax of the Sendmathinfo elementary task is:

```
SENDMATHINFO SIGNALID "SigName" STATUS StatVar
```

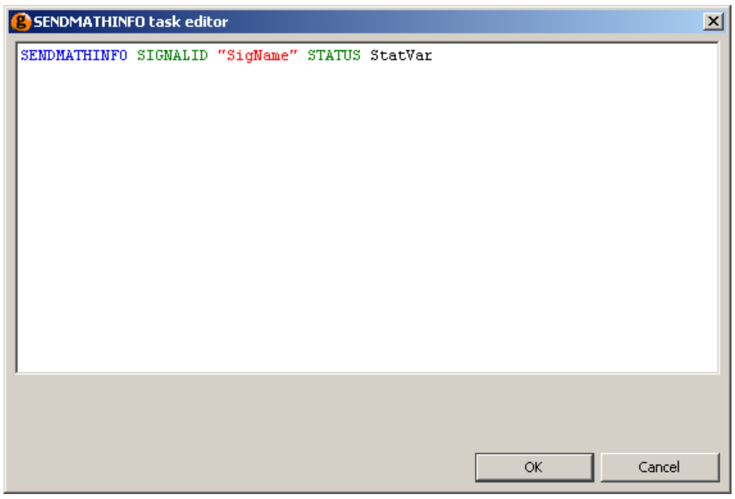
where

SigName	is a string of characters that will be passed to the Foreign Process by gPROMS on executing the Sendmathinfo task. This may be used by the Foreign Process for identifying the actual Sendmathinfo request being executed, which may be useful if there are several Sendmathinfo tasks at different places in the Schedule. The Signalid "*SigName*" specification is optional. An empty string will be passed to the foreign process by default.
StatVar	is a logical *gPROMS* variable that will receive the outcome of the Sendmathinfo request from the Foreign Process. Generally, a value of True will indicate success (or permission to proceed with the simulation), while a value of False may signal failure (or a request to abort the simulation). The Status *StatVar* specification is also optional.

Using Sendmathinfo in the Schedule tab of a Task or Process

As with all other elementary Tasks and user-defined Tasks, the Sendmathinfo Task can be dragged into a Schedule using the Task palette or by right clicking on a hot spot in the Schedule and selecting SendMathInfo from the Add Foreign Process task from the context menu.

When this is done, a dialog will appear so that the Sendmathinfo Task can be configured. Type in the gPROMS language for the Task and press the OK button. The dialog is shown below.



```
SENDMATHINFO SIGNALID "SigName" STATUS StatVar
```

Information transmitted by Sendmathinfo

The mathematical problem that gPROMS is solving over time can be written in the form:

$$f(x, \dot{x}, y, u) = 0$$

where

- *x* is a set of n_x *differential* variables;
- $\dot{x}$ is a set of n_x *time derivatives* of the differential variables *x* (i.e., $\dot{x} \equiv \mathrm{d}x/\mathrm{d}t$);
- *y* is a set of n_y *algebraic* variables;
- *u* is a set of n_u *input* variables;
- $f(\cdot)$ is a set of $n_x + n_y$ ordinary differential (with respect to time) or algebraic equations.

It is worth noting that some of the variables *x, y, u* and the equations *f* may have arisen from the discretisation of variable and equation distributions over one or more DISTRUBUTION_DOMAINs. This is performed by gPROMS automatically *before* the start of the simulation.

At the start of the simulation, the input variables *u* are those which are Assigned in the Process entity. Of the rest of the variables, the ones that occur prefixed with a $ sign at least once in the model are considered to be differential variables *x*, while all others are algebraic variables *y*. However, the classification of a certain variable may change *during* the simulation. For instance:

- Using a Replace task in a Schedule, we may drop a specification imposed on one variable z_1 replacing it by a specification on a *different* variable z_2. In such a case, z_2 will become an input variable, while z_1 will become (most probably) an algebraic variable.
- A discontinuity resulting in a change in the active clause in a conditional (i.e. If/Then/Else or Case) equation in a Model entity may alter the status of one or more variables from differential to algebraic or vice versa.

In any case, the Sendmathinfo task always reports on the state of the model at the time of its execution.

One important characteristic of the mathematical model

$$f(x, \dot{x}, y, u) = 0$$

is that it is usually very *sparse*. This means that each of the equations $f(\cdot) = 0$ involves only a small subset of the variables *x*, $\dot{x}$, *y*, *u*. Since a model may involve tens of thousands of equations and variables, this fact may be of particular significance to a Foreign Process that seeks to carry out some mathematical manipulation of the gPROMS model.

The sparsity of a system of equations is related to the structure of its Jacobian matrix $\boldsymbol{J}$. This is the matrix of the partial derivatives of the equations with respect to the variables, i.e.:

$$\boldsymbol{J}_{ij} \equiv \frac{\partial f_i}{\partial w_j}$$

where *w* is the vector of *all* variables in the system, i.e.:

$$w \equiv \begin{pmatrix} x \\ \dot{x} \\ y \\ u \end{pmatrix}$$

Of course, $\boldsymbol{J}_{ij}$ will be zero if variable w_j does *not* actually occur in equation $f_i = 0$. In practice, the number of *non* zero elements of $\boldsymbol{J}$, n_z, will normally be a very small proportion of the $(n_x + n_y) \times (2n_x + n_y + n_u)$ elements of this matrix. The total number of nonzero elements, as well as the row and column in which each such element occurs, define the *sparsity pattern* of the model. This pattern may change during the simulation as a result of changes in the structure of individual equations (e.g. due to implicit model discontinuities described by If and/or Case constructs in Models. Once again, Sendmathinfo will always report the state of the model at the time of its execution.

In addition to the *SigName* (see also: General syntax), the execution of a Sendmathinfo task transmits to the Foreign Process the information summarised below:

Table 3.4. Timing information

The current simulation time	t

Table 3.5. Overall mathematical model statistics

The total number of variables in the model	$n_x + n_y + n_u$
The total number of equations in the model	$n_x + n_y$
The number of differential variables in the model	n_x
The number of nonzero elements in the equation Jacobian	n_z

Table 3.6. Information on each model variable *x, y, u*

Its current value
The current value of its time derivative (*x* variables only)
Its current classification as "input", "differential" or "algebraic"
Its name (in terms of its full gPROMS pathname)

Table 3.7. Information on the model Jacobian *J*

The row number of each non-zero Jacobian element
The column number of each non-zero Jacobian element
The current numerical value of each non-zero Jacobian element

Side effects of Sendmathinfo

Sendmathinfo tasks have no side effect on the simulation results.

The Linearise elementary task

gPROMS includes a comprehensive model linearisation facility. This provides a means of using non-linear models that are coded in gPROMS and are of arbitrary complexity (e.g. described by mixed, integro-partial differential and algebraic equations) for control-system design using linear-analysis techniques coded, for instance, in standard tools such as MATLAB. A description of this facility is provided here.

The basics of linearisation

gPROMS models typically comprise mixed sets of non-linear differential and algebraic equations. These can be written in the form[2] $f(x, \dot{x}, y, u) = 0$.

[2]Models of distributed processes described by mixed, integro-partial differential and algebraic equations (IPDAEs) are automatically converted to the above form by the gPROMS discretisation techniques.

Here *x(t)* and *y(t)* are the sets of differential and algebraic variables respectively (both of which are unknowns to be determined by the gPROMS simulation) while $\dot{x}(t)$ are the derivatives of *x(t)* with respect to time, *t*. On the other hand, *u* is the set of input variables that are given functions of time[3]Now consider a point $(x^*(t), \dot{x}^*(t), y^*(t), u^*(t))$ on the simulation trajectory that satisfies

$$f(x, \dot{x}, y, u) = 0.$$

By linearising the above equations at this point, we can obtain a linear model of the form:

$$\frac{\partial f}{\partial x}\delta x + \frac{\partial f}{\partial \dot{x}}\delta \dot{x} + \frac{\partial f}{\partial y}\delta y + \frac{\partial f}{\partial u}\delta u = 0,$$

where δz denotes the deviation of the variable z from the reference trajectory $z^*(t)$ and all the partial derivatives are evaluated on the reference trajectory.

For most DAE systems of index 1, the matrix $\left[\frac{\partial f}{\partial \dot{x}} \frac{\partial f}{\partial y}\right]$ is non-singular, and consequently the above system can be re-arranged to the form:

$$\begin{pmatrix} \delta \dot{x} \\ \delta y \end{pmatrix} = -\left[\frac{\partial f}{\partial \dot{x}} \frac{\partial f}{\partial y}\right]^{-1} \left[\frac{\partial f}{\partial x} \frac{\partial f}{\partial u}\right] \begin{pmatrix} \delta x \\ \delta u \end{pmatrix},$$

which is a linearised form of the original non-linear system

$$f(x, \dot{x}, y, u) = 0.$$

The linearised model generated by gPROMS

gPROMS generates a linearised model from three specified items of information:

- the non-linear model described by the equation

  $$f(x, \dot{x}, y, u) = 0.$$

- a set of input variables *U*; and
- a set of output variables *Y*.

The linearised model is of the general form:

$$\delta \dot{X} = A\delta X + B\delta U$$
$$\delta Y = C\delta X + D\delta U$$

where *X* is a set of state variables and *A*, *B*, *C* and *D* are matrices of appropriate dimensions.

Although the above equation seems to be very similar to

$$\begin{pmatrix} \delta \dot{x} \\ \delta y \end{pmatrix} = -\left[\frac{\partial f}{\partial \dot{x}} \frac{\partial f}{\partial y}\right]^{-1} \left[\frac{\partial f}{\partial x} \frac{\partial f}{\partial u}\right] \begin{pmatrix} \delta x \\ \delta u \end{pmatrix},$$

, the variable sets *X*, *Y* and *U* are *not* generally identical to *x*, *y* and *u*, respectively. More specifically:

- the input variables *U* specified by the user are generally a subset of *x, y, u*;
- the output variables *Y* specified by the user are generally a subset of *x, y, u*;

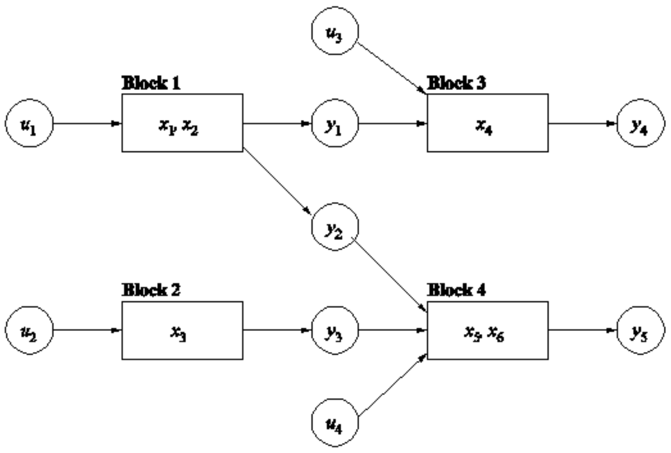
[3]It should be noted that, in gPROMS, the membership of the sets *x(t), y(t), u(t)* may change during the course of the simulation (e.g. via Replace tasks). The analysis presented in this document applies at a particular point during the simulation at which the linearised model is constructed from the nonlinear one.

- the state variables *X* are determined automatically by gPROMS as the minimal subset of *x* that is necessary to express the effects of the specified inputs *U* on the specified outputs *Y* via relationships of the form

$$\delta\dot{X} = A\delta X + B\delta U$$
$$\delta Y = C\delta X + D\delta U$$

To understand the above points, consider a process of the form shown in the figure below . It comprises four inter-connected sub-processes ("blocks").

Figure 3.1. Diagram of process considered.



Now consider the following cases:

Case 1: specify $U \equiv \{u_1, u_2, u_3, u_4\}$ and $Y \equiv \{y_1, y_2, y_3, y_4, y_5\}$.

- This is the "classic" case in which the linearised model

  $$\delta\dot{X} = A\delta X + B\delta U$$
  $$\delta Y = C\delta X + D\delta U$$

  corresponds exactly to the non-linear model

  $$f(x, \dot{x}, y, u) = 0.$$

  The set of states *X* in the linear model is identical to *x*, i.e. $X \equiv \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and the linearised model will be derived from the equations in all four blocks, i.e. the entire non-linear model.

Case 2: specify $U \equiv \{u_1, u_2\}$ and $Y \equiv \{y_4, y_5\}$.

- This is still a "classic" case in the sense that *U* and *Y* are subsets of the true process inputs *u* and outputs *y* respectively. The set of states *X* that is required in the non-linear model

  $$f(x, \dot{x}, y, u) = 0.$$

  to describe the effects of *U* on *Y* still comprises the entire set of differential variables *x*, i.e. $X \equiv \{x_1, x_2, x_3, x_4, x_5, x_6\}$ and the linearised model will still be derived from the entire non-linear model.

Case 3: specify $U \equiv \{u_1, u_3\}$ and $Y \equiv \{y_4\}$.

- This is similar to Case 2. However, the set of states *X* that is required in

  $$\delta\dot{X} = A\delta X + B\delta U$$
  $$\delta Y = C\delta X + D\delta U$$

to describe the effects of *U* on *Y* needs to include only a subset of the differential variables *x*, i.e. $X \equiv \{x_1, x_2, x_4\}$. The linearised model will be derived from the non-linear equations in blocks 1 and 3 only.

Case 4: specify $U \equiv \{y_1\}$ and $Y \equiv \{x_4, y_4\}$.

- This is a "non-classic" case in which the specified set of inputs *U* includes a variable that is actually an output of the non-linear model. The set of states *X* that is required to describe the effects of *U* on *Y* is simply $X \equiv \{x_4\}$. Moreover, only the non-linear equations in block 3 are needed to form the linearised model.

Case 5: specify $U \equiv \{u_2, x_2, u_3\}$ and $Y \equiv \{y_1, y_4, x_6\}$.

- This is also a "non-classic" case in which the specified set of inputs *U* includes a differential variable the non-linear model. The set of states *X* that is required to describe the effects of *U* on *Y* is now $X \equiv \{x_3, x_4, x_5, x_6\}$ and the linearised model is derived from the equations in blocks 2, 3 and 4 of the non-linear model. Note that the variable x_2 appears in the linearised model as an input variable, *U*, and not as a state, *X*. Moreover, since it is a specified input, the equations that determine it (i.e. those in block 1) are not used for generating the linearised model.

Case 6: specify $U \equiv \{u_1, y_2, u_3\}$ and $Y \equiv \{y_4, y_5\}$.

- Here, the specified set of inputs, *U*, includes a variable, y_2, that is actually affected by another element of *U*, namely variable u_1- consequently, y_2 cannot be considered as an independent input variable. In this case, the linearisation procedure automatically removes y_2 from the list of inputs. The corresponding columns of matrices *B* and *D* (which are zeroes) are removed. Therefore, the number of columns of matrices *B* and *D* is reduced accordingly.

Formal statement of the linearisation problem

The six cases presented above illustrate the complexity of what is, at first sight, a rather simple problem. The required analysis is performed automatically by gPROMS and is completely transparent to the user. The overall specification of the new facility is as follows:

Given:

- the non-linear model described by

  $$f(x, \dot{x}, y, u) = 0.$$

- a vector of variable values $x(t), \dot{x}(t), y(t), u(t)$ on the solution trajectory, i.e. satisfying equation

  $$f(x, \dot{x}, y, u) = 0.$$

- a set of input variables *U*; and
- a set of output variables *Y*,

Determine:

- the minimal set of states $X \subseteq \{x\}$; and
- the matrices *A*, *B*, *C* and *D*.

The Linearise elementary task

The linearisation facility is implemented as an elementary task that can be inserted anywhere in a gPROMS Schedule.

The form of the task is illustrated in the example below[4]

```
SCHEDULE
  SEQUENCE
    ...
    ...
    LINEARISE
      SIGNALID "SignalName"
      STATUS StatVar
      INPUT_VARIABLE
        Plant.Reactor.Fin ;
        "InputConcentration" := Plant.Reactor.Xin ;
        Plant.Reactor.Tout ;
      OUTPUT_VARIABLE
        Plant.Column(1).DistillateFlow ;
        Plant.Column(1).DistillatePurity ;
        "BottomFlowrate" := Plant.Column(2).BottomsFlow ;
        Plant.Column(2).BottomsPurity ;
    END
    ...
    ...
  END
```

Note

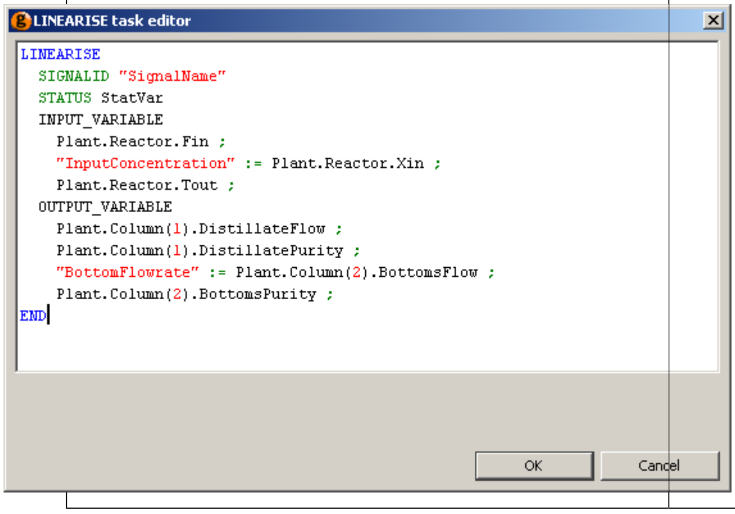
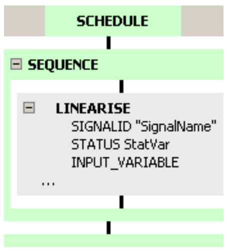
- The alternative spelling LINEARIZE is also allowed.
- The input variable specification begins with the keyword INPUT_VARIABLE. The simpler alternative INPUT is also allowed.
- The output variable specification starts with the keyword OUTPUT_VARIABLE. The simpler alternative OUTPUT is also allowed.
- There must be exactly one input section and one output section. However, they can be specified in any order.
- The SIGNALID and STATUS specifications are optional as in other Foreign Process tasks.
- The LINEARISE task has no side effects on the simulation results.

As with all other elementary Tasks and user-defined Tasks, the Linearise Task can be dragged into a Schedule using the Task palette or by right clicking on a hot spot in the Schedule and selecting Linearise from the Add Foreign Process task from the context menu.

When this is done, a dialog will appear so that the Send Task can be configured. Type in the gPROMS language for the Task and press the OK button. The dialog for the example above is shown below, along with the Schedule.

[4]Note that "tag names" are allowed as aliases for the gPROMS variable path names, and that semicolons are used as separators between different variables in the lists of inputs and outputs. This is consistent with the syntax of other FPI tasks such as Get and Send.

Table 3.8. Example of the Linearise Task

Dialog	In Schedule
	

```
LINEARISE
  SIGNALID "SignalName"
  STATUS StatVar
  INPUT_VARIABLE
    Plant.Reactor.Fin ;
    "InputConcentration" := Plant.Reactor.Xin ;
    Plant.Reactor.Tout ;
  OUTPUT_VARIABLE
    Plant.Column(1).DistillateFlow ;
    Plant.Column(1).DistillatePurity ;
    "BottomFlowrate" := Plant.Column(2).BottomsFlow ;
    Plant.Column(2).BottomsPurity ;
END
```

```
LINEARISE
  SIGNALID "SignalName"
  STATUS StatVar
  INPUT_VARIABLE
...
```

The gFPLINEARISE procedure

The Linearise task ultimately results in a call to a new Foreign Process Interface (FPI) routine, gFPLINEARISE.

This complements the existing FPI routines, gFPI, gFPPAUSE, gFPGET, gFPSEND, gFPSENDM and gFPT.

gFPLINEARISE receives the following list of arguments from gPROMS:

FPID	Full name of Foreign Process.
FPHANDLE	Identifier value assigned by gFPI.
PRNAME	Name of executing gPROMS Process.
SIGNAL	SignalName string specified for current instance of the task in the Schedule.
TIME	Current simulation time.
NU	Number of input variables specified in the Linearise task. Any input variables that are not independent (*cf.* Case 6) will be removed, so in this case, this argument will give the number of independent input variables.
UINDICES	Indices of input variables in the global variable array. Any input variables that are not independent (*cf.* Case 6) will be removed, so in this case, this argument will give the indices of independent input variables.
UNAMES	Names (tag names) of input variables. Any input variables that are not independent (*cf.* Case 6) will be removed, so in this case, this argument will give the names of independent input variables.
NY	Number of output variables specified in the Linearise task.
YINDICES	Indices of output variables in the global variable array.
YNAMES	Names (tag names) of output variables.
NX	Number of the minimal subset of state variables.
XINDICES	Indices of state variables in the global variable array.
XNAMES	Names of state variables.
A	Matrix *A* in

	$\delta\dot{X} = A\delta X + B\delta U$ $\delta Y = C\delta X + D\delta U$ (NX $\times$ NX elements, sorted by rows).
B	Matrix *B* in $\delta\dot{X} = A\delta X + B\delta U$ $\delta Y = C\delta X + D\delta U$ (NX $\times$ NU elements, sorted by rows).
C	Matrix *C* in $\delta\dot{X} = A\delta X + B\delta U$ $\delta Y = C\delta X + D\delta U$ (NY $\times$ NX elements, sorted by rows).
D	Matrix *D* in $\delta\dot{X} = A\delta X + B\delta U$ $\delta Y = C\delta X + D\delta U$ (NY $\times$ NU elements, sorted by rows).
STATUS	Flag indicating success/failure of the Linearise task: 1: successful execution; 0: non-existent Jacobian element; -1: structurally singular DAE system; -2: failure of linear system solver.

All of the above arguments are already initialised with the corresponding information by gPROMS on entry to gFPLINEARISE. They must be left unchanged on exit.

If implemented in FORTRAN, gFPLINEARISE will be a subroutine with the following declaration:

```
 SUBROUTINE gFPLINEARISE(FPID, FPHANDLE, PRNAME, SIGNAL, TIME,
+                        NU, UINDICES, UNAMES,
+                        NY, YINDICES, YNAMES,
+                        NX, XINDICES, XNAMES,
+                        A, B, C, D,
+                        STATUS)

 INTEGER FPHANDLE, NU, NY, NX, STATUS
 INTEGER UINDICES(NU), YINDICES(NY), XINDICES(NX)
 CHARACTER * 256 FPID, PRNAME, SIGNAL
 CHARACTER * 256 UNAMES(NU), YNAMES(NY), XNAMES(NX)
 DOUBLE PRECISION TIME
 DOUBLE PRECISION A(NX * NX), B(NX * NU), C(NY * NX), D(NY * NU)
```

If implemented in C/C++, the gFPLINEARISE function will have the following declaration:

```
 extern "C" void gfplinearise_(
                  const char *foreignProcessId,
                  const long *foreignProcessHandle,
                  const char *processName,
                  const char *taskSignalName,
                  const double *currentTime,
                  const unsigned long *numberOfInputVariables,
                  const long *indicesOfInputVariables,
                  const char (*namesOfInputVariables)[256],
                  const unsigned long *numberOfOutputVariables,
                  const long *indicesOfOutputVariables,
                  const (*namesOfOutputVariables)[256],
                  Const unsigned long *numberOfMinimalStateVariables,
                  Const long *indicesOfMinimalStateVariables,
                  const (*namesOfMinimalStateVariables)[256],
                  Const double *aMatrix,
                  Const double *bMatrix,
                  Const double *cMatrix,
                  Const double *dMatrix,
                  long *Status);
```

The default gFPLINEARISE procedure

gPROMS is provided with a default FPI implementation.

Each execution of the default implementation of the gFPLINEARISE routine creates a file called *PrName_n*.gLINEAR, where:

- *PrName* is the name of the process that is currently executed (as passed to gFPLINEARISE by gPROMS).
- *n* is the number of instances of the Linearise task that have been executed so far in the current simulation (e.g. 1, 2, 3, *etc*).

The above files are placed in the Results folder of the corresponding Case. A sample file is shown below.

```
gPROMS Linearisation Utility
============================

Output generated at 11:06 on 2001-01-25

  Process : TEST_LINEARMODEL
Signal ID : LinearisationTest

Linearised model generated at simulation time 0 in the form:

    dX/dt = A X + B U
      Y   = C X + D U

Number of input   variables, U :       2
Number of output variables, Y :       2
Number of state   variables, X :       5

Input variables, U
------------------
  Index  Name
  ~~~~~  ~~~~
      4  UNIT_1.U1
      8  UNIT_1.U3

Output variables, Y
-------------------
  Index  Name
  ~~~~~  ~~~~
     14  UNIT_1.Y(4)
     15  UNIT_1.Y(5)

State variables, X
```

```
------------------
  Index  Name
  ~~~~~  ~~~~
      1  UNIT_1.X_1
      2  UNIT_1.X_2
      5  UNIT_1.X_4
      7  UNIT_1.X_5
      9  UNIT_1.X_6

Matrix A
~~~~~~~~
A = [
        -1.6711     0.057895         0.0000       0.0000       0.0000
    -0.073810     -0.39246          0.0000       0.0000       0.0000
      -0.24211     -0.30080    0.0023256       0.0000       0.0000
        -3.8452       -2.2786          0.0000      -41.131      -19.635
        -1.0658     -0.63158          0.0000      -18.403      -8.3281
      ]

Matrix B
~~~~~~~~
B = [
        1.5605          0.0000
     -0.70119          0.0000
     -0.68231    -0.033915
       -2.1362          0.0000
     -0.59211          0.0000
     ]

Matrix C
~~~~~~~~
C = [
     -0.61517     -0.76430     -0.39535       0.0000       0.0000
        0.0000        0.0000        0.0000      -42.254      -20.028
      ]

Matrix D
~~~~~~~~
D = [
      -1.7337     -0.15116
       0.0000       0.0000
      ]
```

Using the FPI

Once you introduce one or more elementary communication tasks in your simulation Schedule, gPROMS will seek to communicate with an external software package during the simulation. However, some additional information is required to specify exactly *which* external piece of software is to be involved in this communication. This issue is discussed here.

The standard FPI implementation

The standard gPROMS executable installed on your system already provides a basic FPI implementation. In this implementation, the elementary communication tasks other than Linearise operate in the following manner:

- The Pause task prints a message to the screen and pauses the simulation until the user presses the ENTER (or carriage return) key.

- The Get task prints a message to the screen; then the current value of each of the variables being communicated is printed to the screen and the user is requested to enter a new value.
- The Send task prints a message to the screen followed by the current values of the variables being communicated.
- The Sendmathinfo task prints a message to the screen followed by all the information normally transmitted by this task.

Albeit quite basic, the standard FPI implementation is often useful for initial testing of input files involving communication tasks, and, in particular, for ensuring that the correct communication actions are initiated by a given simulation.

The eventFPI implementation of the Foreign Process Interface

The eventFPI is provided for use within ModelBuilder as an alternative to the default implementation of the Foreign Process Interface . The eventFPI makes use of a set of dialogs that appear automatically during the execution of a dynamic simulation activity whenever an elementary FPI task is encountered in the Schedule.

The eventFPI provides an easy-to-implement way of allowing the user of the Model to communicate with the dynamic simulation in a simple and convenient manner. It also allows the use of the ModelBuilder for testing event-based FPI-based communication in models that are ultimately destined to be exported for use within gPROMS-based Applications[5].

To use the eventFPI, simply specify the value of the FPI parameter in Solutionparameters section of the Process entity to `"eventFPI"`, as shown below:

```
SOLUTIONPARAMETERS
        FPI := “eventFPI” ;
```

The eventFPI interaction dialogs

All eventFPI dialogs include:

- the SIGNAL ID identifying the elementary task being executed;
- the simulation time at which the interaction occurs;
- the value of the status of the execution of this elementary task;
  - For GET, SEND and PAUSE elementary tasks, the status is always indicated as SUCCESS on entry. The user may change it to FAIL via the drop-down menu provided. It should be noted that this simply sets the value of the STATUS logical flag associated with this elementary task (e.g. StatFlag in the first figure below) to FALSE; however, it does not automatically stop the simulation. It is up to the model developer to intercept this flag and take appropriate action in the simulation SCHEDULE.
  - For SENDMATHINFO and LINEARISE tasks, the status has an appropriate value on entry indicating the status of the execution of this task[6]. If this value is anything other than SUCCESS when the user presses the OK button, the simulation will terminate immediately.
- an OK button that allows the simulation to continue, subject to the value of the status (see above);
- an ABORT button that terminates the simulation immediately.

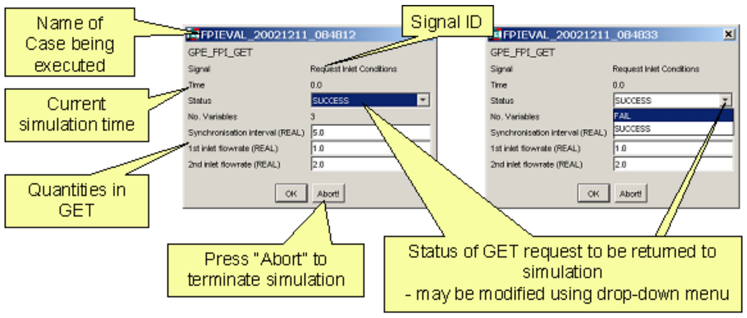
The dialogs provided by eventFPI are illustrated in detail for the case of the GET elementary task in the figure below, and more briefly for the PAUSE, SEND, SENDMATHINFO and LINEARISE tasks in the subsequent figures.

[5]gPROMS-based applications (gBAs) are independent software programs that make use of the gPROMS engine (“server”) and one or more gPROMS Models. For a full description, see the gBA Developer Guide.
[6]E.g., in the case of LINEARISE, a failure that has occurred in constructing the linearised state-space model.

Table 3.9. Example of elementary GET task in eventFPI

```
GET
  SIGNALID "Request Inlet Conditions"
  STATUS StatFlag ;
  SimInterval := "Synchronisation interval" ;
  Tank.Fin(1) := "1st inlet flowrate"       ;
  Tank.Fin(2) := "2nd inlet flowrate"       ;
END
```

(a) gPROMS language


Name of Case being executed
Signal ID
Current simulation time
Quantities in GET
Press "Abort" to terminate simulation
Status of GET request to be returned to simulation - may be modified using drop-down menu

FPIEVAL_20021211_084812

GPE_FPI_GET

Signal	Request Inlet Conditions
Time	0.0
Status	SUCCESS
No. Variables	3
Synchronisation interval (REAL)	5.0
1st inlet flowrate (REAL)	1.0
2nd inlet flowrate (REAL)	2.0

OK Abort!

FPIEVAL_20021211_084833

GPE_FPI_GET

Signal	Request Inlet Conditions
Time	0.0
Status	SUCCESS (FAIL / SUCCESS)
No. Variables	
Synchronisation interval (REAL)	
1st inlet flowrate (REAL)	1.0
2nd inlet flowrate (REAL)	2.0

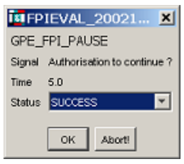
OK Abort!

(b) Interaction dialog

Table 3.10. Example of elementary PAUSE task in eventFPI

```
PAUSE
  SIGNALID "Authorisation to continue?"
  STATUS StatFlag
```

(a) gPROMS language



FPIEVAL_20021...

GPE_FPI_PAUSE

Signal	Authorisation to continue ?
Time	5.0
Status	SUCCESS

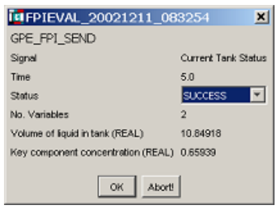
OK Abort!

(b) Interaction dialog

Table 3.11. Example of elementary SEND task in eventFPI

```
SEND
  SIGNALID "Current Tank Status"
  STATUS StatFlag
  "Volume of liquid in tank"      := Tank.V     ;
  "Key component concentration"  := Tank.C(1) ;
END
```

(a) gPROMS language



FPIEVAL_20021211_083254

GPE_FPI_SEND

Signal	Current Tank Status
Time	5.0
Status	SUCCESS
No. Variables	2
Volume of liquid in tank (REAL)	10.84918
Key component concentration (REAL)	0.65939

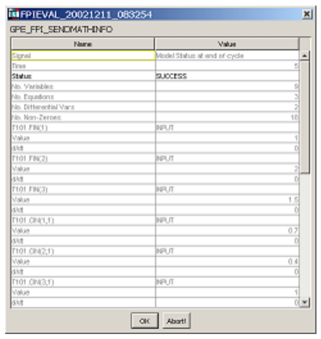
OK Abort!

(b) Interaction dialog

Table 3.12. Example of elementary SENDMATHINFO task in eventFPI

```
SENDMATHINFO
  SIGNALID "Model status at end of cycle"
  STATUS StatFlag
```

(a) gPROMS language



FPIEVAL_20021211_083254

GPE_FPI_SENDMATHINFO

Name	Value
Signal	Model Status at end of cycle
Time	5
Status	SUCCESS
No. Variables	9
No. Equations	3
No. Differential Vars	2
No. Non-Zeroes	10
T101.FIN(1)	INPUT
Value	1
d/dt	0
T101.FIN(2)	INPUT
Value	2
d/dt	0
T101.FIN(3)	INPUT
Value	1.5
d/dt	0
T101.CIN(1,1)	INPUT
Value	0.7
d/dt	0
T101.CIN(2,1)	INPUT
Value	0.4
d/dt	0
T101.CIN(3,1)	INPUT
Value	1
d/dt	0

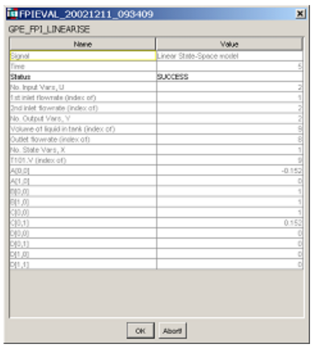
OK Abort!

(b) Interaction dialog

Table 3.13. Example of elementary LINEARISE task in eventFPI

```
LINEARISE
  SIGNALID "Linear State-Space model"
  INPUT
    "1st inlet flowrate" := Tank.Fin(1) ;
    "2nd inlet flowrate" := Tank.Fin(2) ;
  OUTPUT
    "Volume of liquid in tank" := Tank.V ;
    "Outlet flowrate"          := Tank.F ;
END
```

(a) gPROMS language



Name	Value
Signal	Linear State-Space model
Time	5
Status	SUCCESS
No. Input Vars, U	2
1st inlet flowrate (index of)	1
2nd inlet flowrate (index of)	2
No. Output Vars, Y	2
Volume of liquid in tank (index of)	9
Outlet flowrate (index of)	8
No. State Vars, X	1
T101.V (index of)	9
A[0,0]	-0.152
A[1,0]	0
B[0,0]	1
B[1,0]	1
C[0,0]	1
C[0,1]	0.152
D[0,0]	0
D[0,1]	0
D[1,0]	0
D[1,1]	0

(b) Interaction dialog

Using non-standard FPI implementations

In most practical applications of the FPI, you will eventually want to use an FPI implementation that is more powerful than the standard one provided by gPROMS (see also: standard FPI implementation). This implementation will typically have been developed either by you or by another person (e.g. a gPROMS system programmer) and will be written as code in a conventional programming language (e.g. FORTRAN or C). The precise form of this code is discussed in detail in the chapter entitled "Developing New FPI Implementations" in the "gPROMS System Programmer Guide".

Here, we will assume that the FPI implementation that we wish to use is already embodied in:

- a shared object library (.so) file under the UNIX operating system, or,
- a dynamic link library (.dll) file under the MS Windows NT operating system.

For the purposes of illustration, let us suppose that this FPI implementation file is called MyFPI.so or MyFPI.dll.

For gPROMS to be able to use the above FPI implementation to execute any communication tasks that you may have in your simulation Schedule, you need to ensure two things: first, that gPROMS knows which FPI implementation you intend to use; and secondly, that it is able to find the corresponding .so or .dll for this implementation. We deal with each of these issues separately below.

Specifying the FPI implementation to be used

The FPI implementation to be used by a given simulation must be specified in the Solutionparameters section of the corresponding Process entity. The parameter that needs to be specified is called FPI, and the name of the FPI implementation is enclosed in quotes. For instance, the specification:

```
  SOLUTIONPARAMETERS
         FPI := "MyFPI" ;
```

indicates that an FPI implementation called MyFPI[7]

should be used for the current simulation.

Some FPI implementations may require additional data that determine their precise behaviour. Typically, such data will be held in an input file but this is not necessarily so. In any case, the additional data define a specific "instance"

[7]This would be implemented as either MyFPI.so under UNIX or MyFPI.dll under MS Windows NT.

of an FPI implementation. The name of this instance must be appended to that of the FPI implementation[8] and the two must be separated with a double colon (::) separator.

For instance, suppose that the FPI implementation MyFPI needs to operate at some constant factor to real-time, and that this factor (together with, perhaps, other parameters) must be specified in a text file that you need to provide. Now, suppose that you have prepared such a file and you have called it /home/user01/MyFPI.dat. In this case, the FPI specification will take the form:

```
SOLUTIONPARAMETERS
      FPI := "MyFPI::/home/user01/MyFPI.dat" ;
```

We note that gPROMS always passes the complete FPI specification string to the FPI implementation. It is up to the FPI implementation to break down this string into its class and instance names (if appropriate), and to decide what to do with them.

A typical application of this facility is in the case of the ExcelFP FPI implementation that provides a link between gPROMS simulations and user interfaces implemented in Microsoft Excel spreadsheets (see also: Microsoft Excel Foreign Object and Foreign Process Interfaces). In such cases, the above instance name is simply the name of the Excel spreadsheet to be used for this purpose.

Locating the FPI implementation file

Once you have specified the FPI implementation that you wish to use in a given simulation (see also: specifying the FPI implementation), you must ensure that gPROMS is able to locate the corresponding FPI implementation file. This can be done by locating this file at an appropriate place in your file system.

If an *absolute* path name is specified for the FPI implementation in the Solutionparameters section of the Process (see also: specifying the FPI implementation), then gPROMS will look for this specific file. For instance, the specification:

```
SOLUTIONPARAMETERS
     FPI := "/home/user01/MyFiles/MyFPI" ;
```

on a UNIX system will cause gPROMS to look for a file called /home/user01/MyFiles/MyFPI.so. If this file is not found, the simulation will be aborted.

If an absolute path name is *not* specified for the FPI implementation in Solutionparameters, then gPROMS will search for the FPI implementation file at three different locations:

1. In a subdirectory called "fpi" of the export directory, see the chapter entitled "gPROMS ModelBuilder" in the "gPROMS ModelBuilder User Guide". For this to be done the FPI implementation file must be imported or linked to ModelBuilder.
2. In a directory specified by the operating system "environment variable" GPROMSFPIDIR if the latter has been given a value[9]
3. In sub-directory "fpi" of the gPROMS installation directory[10]

The three directories are searched in the order listed above. If the FPI implementation file is found at one of these locations, then the search terminates successfully and gPROMS proceeds to use this file. Otherwise, the simulation is aborted.

[8]In object-oriented programming terminology, the FPI implementation is called the "class" of all its instances.
[9]The manner in which UNIX environment variables can be given a value depends on the command line interpreter ("shell") being used. For instance, to set GPROMSFPIDIR to a directory called /home/ps/library/FPI on a UNIX system operating under the C shell, type the command:

```
setenv GPROMSFPIDIR /home/library/FPI
```

The equivalent command under the Korn and Bourne shells is:

```
GPROMSFPIDIR=/home/library/FPI; export GPROMSFPIDIR
```

Please note that there are no blank spaces on either side of the equals sign in the above command.
[10]On UNIX systems, the gPROMS installation directory is normally specified by the GPROMSHOME environment variable.

FPI performance issues

The issue of FPI performance is particularly important in view of the fact that many of its applications will operate in real-time under stringent timing constraints.

Some of the factors that may affect this performance are considered below.

Effects of model size

In general, the larger the model, the longer its solution will take! In many cases, of course, the size of the model will be determined by the application, and there may be little scope for reducing it. However, for models that are going to be used time and time again in critical on-line applications, it is worth investing some effort in this direction.

In particular, large models arising from the discretisation of distributed systems may be reduced by selecting a smaller number of discretisation elements while taking care to maintain sufficient accuracy for the purposes of the application-this accuracy can be relatively low for many operator training applications.

You may also try to eliminate unnecessary intermediate variables and the equations defining them from such models. For instance, the original model for a tubular reactor unit may include variables for the reaction constants and the reaction rates, all of which are distributed over one or more spatial dimensions. Although this may be advantageous from the point of view of clarity at the model development stage, it may be causing unnecessary overheads during the solution. In such cases, it may be better to eliminate these variables and equations by substituting the kinetic and reaction rate expressions directly within the material and energy conservation equations.

Effects of model re-initialisation

As explained in: Side effects of Get, the interaction of a gPROMS Process with the Foreign Process through a Get task may cause changes to the underlying model. This then leads to the need for re-initialisation, i.e. a calculation aiming at re-establishing consistent values for all variables in the model.

In general, this re-initialisation calculation involves the solution of the model equations, and can, therefore, be very time consuming. Fortunately, it takes place after Get tasks *if and only if* the value of at least one of the input or differential variables involved is changed by the Foreign Process. In some cases, it may be possible to exploit this fact by the FPI implementation not passing back to gPROMS insignificantly small changes.

Effects of results logging

By default, gPROMS records the values of all the variables in the model at each reporting interval, as specified either in the Solutionparameters part of the Process in the input file (see gPROMS ModelBuilder User Guide):

```
# PROCESS Example
  UNIT Plant AS BigModel
  SET
    ...
  ASSIGN
    ...
  PRESET
    ...
  INITIAL
    ...
  SOLUTIONPARAMETERS
    REPORTINGINTERVAL := 10 ;
  SCHEDULE
    ...
```

or manually at the start of the Process execution:

```
Executing process Example...

gPROMS simulation initialisation took 0.010 seconds.

All 54 variables will be monitored during this simulation!

gPROMS problem construction took 0.020 seconds.

What reporting interval do you require?
Specify time interval as a positive number
Enter Value :
 ....
```

In both cases, the above specifications will result in values for all variables that are currently being Monitored (see gPROMS ModelBuilder User Guide) to be transmitted to all Output Channels that are currently active (see gPROMS ModelBuilder User Guide) for archiving and display every 10 time units. For simulations involving large models, the effect of the associated overhead on real-time performance may well be unacceptable. It may also be unnecessary as the Foreign Process may itself wish to assume responsibility for the archiving and display of the important variables transmitted to it from gPROMS via Send tasks.

There are two easy ways to solve this problem. The first is to turn off all output channels in the Solutionparameters section of the Process (see gPROMS ModelBuilder User Guide):

```
SOLUTIONPARAMETERS
  gRMS         := OFF ;
  gPLOT        := OFF ;
  gExcelOutput := OFF ;
  gUserOutput  := OFF ;
```

The second is to switch off variable monitoring and/or to restrict the number of variables sent to the output channel. Monitoring can be turned off either before the simulation starts by setting:

```
Monitor := Off ;
```

in the Solutionparameters section or during the simulation by using the Monitor task:

```
SCHEDULE
  SEQUENCE
    ...
    MONITOR OFF
    ...
  END
```

If there are many intermediate variables that do not need to be monitored, they can be excluded from the set of variables that is sent to the output channel. This is done by specifying only the variables of interest in the Monitor section of the Process (see gPROMS ModelBuilder User Guide).

Effects of FPI communication overheads

The communication between gPROMS and the FPI procedures is done using standard procedure calls, and usually incurs negligible costs. However, this may not be the case if the FPI routines then proceed to communicate with some other software through a different mechanism involving data transmission across networks (e.g. using UNIX "sockets" or some other message-passing protocol).

There is also some overhead incurred by gPROMS just before and just after calling the FPI procedures. This is mainly due to the need for copying data between the arrays to be passed to FPI and the internal gPROMS data structures. This type of cost is likely to be significant only for Sendmathinfo tasks applied to large models.

Chapter 4. Microsoft Excel Foreign Object and Foreign Process Interfaces

Introduction

The Microsoft Excel™[1]Foreign Object and Foreign Process interfaces are designed to allow gPROMS to interact dynamically with calculations performed in Microsoft Excel. The Microsoft Excel Foreign Object interface can be used to provide values for Parameters and external calculations for Variables in a gPROMS simulation (e.g. to link a gPROMS model with an accounting spreadsheet). The Microsoft Excel Foreign Process interface can be used for a number of purposes:

- An Excel "front-end" to a gPROMS simulation can be created where the user can interactively provide input to the simulation and have the results update the spreadsheet automatically.
- The results of a gPROMS simulation can be tabulated in an Excel spreadsheet, or tabulated data in a spreadsheet can be used, over time, to update variables in a simulation.
- The Microsoft Excel FPI can be used to synchronise a gPROMS simulation with external software.

In this chapter, we describe how to use Microsoft Excel Foreign Objects and the Foreign Process interface from gPROMS and how to create and use foreign objects and the Foreign Process interface in Microsoft Excel.

Using Microsoft Excel Foreign Objects

Microsoft Excel Foreign Objects are specified in the gPROMS input file in the same way that other Foreign Objects are.

In the gPROMS Model, a parameter must be defined as Foreign_object with the class "ExcelFO":

```
  PARAMETER
    ParameterPath       AS FOREIGN_OBJECT "ExcelFO"
```

The file that will be used by ExcelFO must then be specified in a Set statement:

```
  SET
    ParameterPath       := "ForeignObjectValue" ;
```

In this case, *ForeignObjectValue* must be the full path name of the file (recommended) or the file must be present in the gPROMS export directory.

This Foreign Object can be used in equations to provide external calculations. For example, imagine that a workbook with the filename Capital_Costs.xls calculates the approximate capital cost of various units. These calculations can be defined as methods of this workbook, with inputs such as the volume of the unit. We might then have:

```
PARAMETER
    CapCosts       AS FOREIGN_OBJECT "ExcelFO"
    ...

  SET
    CapCosts       := "Capital_Costs.xls"
    ...
```

[1]The facilities described in this Appendix are supported by Microsoft Excel 97 and later versions.

```
EQUATION
  Tank_Cost = CapCosts.Tank(Tank_Volume) ;
  ...
```

In the example above, we calculate the capital cost of a tank, based on its volume using the Tank method in the Capital_Costs.xls workbook (resident in the gPROMS working directory).

In Creating Microsoft Excel™ Foreign Objects, we describe in detail how to create Foreign Object methods in Microsoft Excel workbooks.

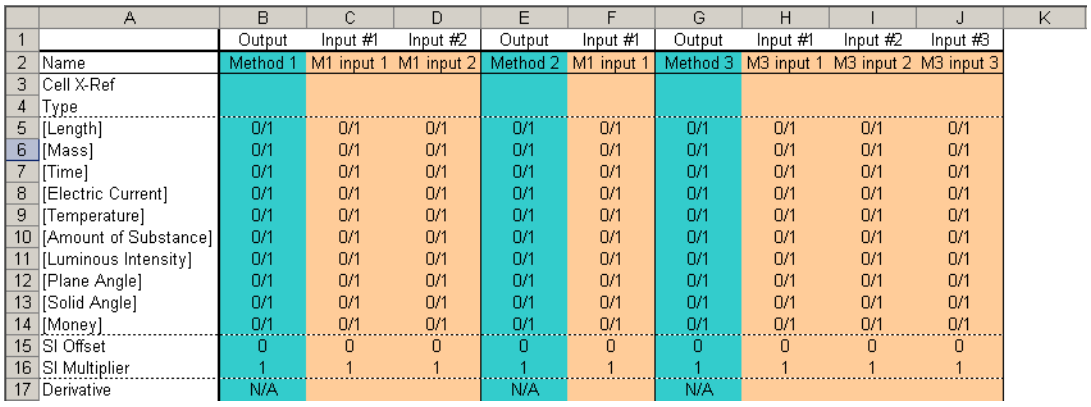
Creating Microsoft Excel Foreign Objects

Microsoft Excel Foreign Objects are created from Excel workbooks by including method definition worksheets. The method definition worksheets specify, for each method:

- the method name;
- which cells are to provide the output values of the method;
- the names of any inputs;
- which cells correspond to the inputs;
- the parameter type (Real, Integer, or Logical);
- information for consistency checking of dimensions;[2]
- cells that provide values for the derivative of the output with respect to the input (if available).

Worksheets that contain method definitions must have names that begin with gFO- (e.g. gFO-Scalars and gFO-Vectors), and there may be any number of method definition worksheets. The format of the worksheet is shown in the figure below.

Figure 4.1. Format of the method definition worksheet.



	A	B	C	D	E	F	G	H	I	J	K
1		Output	Input #1	Input #2	Output	Input #1	Output	Input #1	Input #2	Input #3	
2	Name	Method 1	M1 input 1	M1 input 2	Method 2	M1 input 1	Method 3	M3 input 1	M3 input 2	M3 input 3	
3	Cell X-Ref										
4	Type										
5	[Length]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
6	[Mass]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
7	[Time]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
8	[Electric Current]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
9	[Temperature]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
10	[Amount of Substance]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
11	[Luminous Intensity]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
12	[Plane Angle]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
13	[Solid Angle]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
14	[Money]	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1	
15	SI Offset	0	0	0	0	0	0	0	0	0	
16	SI Multiplier	1	1	1	1	1	1	1	1	1	
17	Derivative	N/A			N/A		N/A				

Multiple method definitions can exist side-by-side on a worksheet. *ExcelFO* determines where a method definition starts by searching for the word "Output" in the first row. The first method definition must start in column B; column A is ignored so that the entries above can be used for guidance. Any subsequent column in row 1 that contains anything other than "Output" is treated as an input for the current method. If *ExcelFO* finds an empty cell while scanning the first row (other than A1), it stops searching for any more method definitions on the current worksheet and moves on to the next method-definition worksheet (method-definition worksheets are processed in order from left to right). Therefore the method definitions must not be separated by blank columns; any definitions to the right of a blank column will be ignored.

[2]These are for future compatibility with new versions of gPROMS.

In the figure above, there are three methods defined: Method 1, Method 2 and Method 3. The two inputs for Method 1 must be in columns C and D. Column E must then contain the definition of the output for Method 2; if this column were left blank, then all of the definitions to the right would be ignored. When *ExcelFO* gets to column K, it stops processing methods: any information to the right of column K will be ignored.

The most convenient way to set up the method definitions is to use the Method Definition Macro. Although the cross references and output types can be added using the macro, the worksheet will need to be edited by hand to specify the dimensions of the inputs and outputs. When doing so, make sure not to move any columns.

Below each entry in row 1, *ExcelFO* searches for information in the following order:

row 2

- The method name (if the column begins with the keyword "Output") or the input name must be specified in row 2. Note that since gPROMS is not case sensitive, the methods method1 and Method1 are considered identical. If there are multiple definitions of the same method in your workbook, the first definition that gPROMS finds will be used; all subsequent definitions will be ignored and a warning message will be displayed when the gPROMS input file is executed.

row 3

- The third row contains cell cross-references for the inputs and output of the method. This is described in more detail below.

row 4

- The variable or parameter type must be specified here. The entries in this row must correspond with the type of the gPROMS Parameter that is being transmitted (i.e. Real, Integer or Logical) or be set to "Real" for gPROMS Variables. *ExcelFO* only recognises the first letter of the entry and is case insensitive.

rows 5-16

- The information for checking the consistency of dimensions is specified here. Rows 5 to 14 contain the indices of the dimensions in rational form. For example, for a parameter with dimensions of [Length]^2 the string "2/1" would be entered in row 5; if the units of a parameter were kg/m^3, the strings "-3/1" and "1/1" would be entered in rows 5 and 6 respectively. The indices of the dimensions must be entered in the order [Length], [Mass], [Time], [Electric current], [Temperature], [Amount of substance], [Luminous intensity], [Plane angle], [Solid angle], [Money]. Rows 15 and 16 contain the *offset* and *multiplier* required to convert the units of the parameter (or output) to SI using the equation: $SIunits = offset + multiplier \times nonSIunits$. For example, if the units of a parameter were Celsius, the *offset* and *multiplier* would be specified as 273.15 and 1 respectively.

row 17

- Contains optional cell references for the derivative of the method output with respect to its inputs. This is ignored for columns that specify outputs. Blank cells indicate that the derivative is unavailable.

Specifying cell cross-references

Cell cross-references are specified as is normal in Excel: i.e. by typing "=<sheetname>!<range>" to specify a range from a particular worksheet. This can be done automatically in Excel by pressing "=" then selecting the worksheet with the mouse, highlighting the range and pressing the RETURN or TAB key.

It is important to make sure that the number of cells specified in the range is equal to the number of elements of the variable or parameter in gPROMS. For variables and parameters that are scalars or vectors, the specification of the ranges is trivial; vectors may be specified in columns or rows. For variables and parameters of higher dimension, the range must be specified in a specific manner. For two-dimensional arrays, the range of cells must be a rectangle with the number of rows equal to the number of elements in the first index and the number of columns equal to the number of elements in the second index. The example below illustrates this more clearly.

Consider a parameter defined by the following:

```
PARAMETER
    A1         AS ARRAY(2,3) OF REAL
```

The array of cells in the Excel worksheet that correspond to this should be a rectangle three columns wide and two rows deep. It is important to check that this is specified correctly because gPROMS only checks that the number of elements is correct. The first figure below shows how the elements of the parameter are related to the array of cells in Excel; the second figure below illustrates how the elements will be assigned incorrectly if the cells are specified incorrectly.

Correspondence of elements in a 2×3 array with correctly and incorrectly specified cell blocks in Excel.

Table 4.1. Cell range specified correctly

A1(1,1)	A1(1,2)	A1(1,3)
A1(2,1)	A1(2,2)	A1(2,3)

Table 4.2. Cell range specified incorrectly

A1(1,1)	A1(1,2)
A1(1,3)	A1(2,1)
A1(2,2)	A1(2,3)

Although it is not envisaged that arrays with more than two dimensions will be necessary, support for higher dimensions is possible and is outlined in: Cell ranges for arrays with more than two dimensions.

Using the method definition macro

Your ExcelFO installation comes with a template workbook that contains macros for generating method definition worksheets and foreign process worksheets. To make use of the macros, you must have the file gXLmacros.xls open while working on your Excel workbook. To run the method definition macro select *Tools* menu, then select *Macro*, then *Macros...* (or just press ALT+F8), select the MakeFOsheet macro and left-click on the "Run" button.

You will now be presented with a dialogue box that contains a list of current macro definition worksheets and four buttons:

"New...'

- Left-click on this button to create a new method definition worksheet.

"Delete''

- Select a method definition worksheet from the list and left-click this button to delete it.

"Methods...''

- Select a method definition worksheet from the list and left-click this button to edit, add or delete methods from the worksheet.

"Exit''

- Left-click to return to the workbook. Any modifications you made during the session were saved automatically.

When you click on the "New..." button, you will be prompted to enter the name of the new worksheet. Enter the name *without* the gFO- prefix; this is added automatically. Once there is a method definition worksheet, you can select it and click on the "Methods..." button. This takes you to the method dialogue, where new methods can be added and existing methods can be edited or deleted.

To create a new method, type the name in the space provided and press the "Add" button. You may also specify the type (which must be Real) and the cell cross-reference. Note that you must include the "=" when typing in the cross-reference and that the worksheet must be specified along with the cell range. If you do not specify the type and cell cross-reference, then they can be edited in the workbook later.

Once a method is defined, you may add, remove or edit its inputs by clicking on the "Inputs >>" button, below the list of methods. The input dialogue behaves exactly as the methods dialogue. To return to the methods dialogue, click on the "<< Methods" button, below the list of inputs.

Once you have finished adding methods and inputs to the worksheet, click on the "OK" button. If you wish to discard any edits, press the "Cancel" button.

Note that you can edit most properties by using the macro, but it is not possible to edit the names of worksheets, methods or inputs. However, these are easily modified in the workbook.

To complete the method definition worksheet, you need to edit the workbook directly to include cross-references for derivatives, to specify the information on dimensions (if necessary) and to complete the specification of input and output cross-referencing.

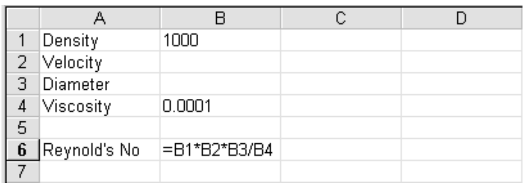
A simple example

To illustrate many of the features described above, we consider the trivial example of calculating the Reynold's Number for fluid flow in a pipe. The listing below shows excerpts from a Model where the Reynolds number is calculated using the *ExcelFO* Foreign Object interface. The physical properties of the fluid are also obtained from the Microsoft Excel Foreign Object.

```
PARAMETER
    FO                AS FOREIGN_OBJECT "ExcelFO"
    Diameter          AS REAL
    Viscosity         AS REAL
    Density           AS REAL
    ...

  VARIABLE
    Velocity          AS Flow
    Re                AS Dimensionless
    ...

  SET
    FO                := "FlowCalc.xls"   ;
    Diameter          := 0.1              ;
    Viscosity         := FO.Viscosity     ;
    Density           := FO.Density       ;

  EQUATION
    ...

    Re = FO.ReNumber(Velocity,Diameter) ;

    ...
```

To make the Foreign Object, we need to open a new file called FlowCalc.xls in Microsoft Excel and enter the Reynold's Number calculation and the method definitions. The worksheet, named Sheet1, containing the Reynold's Number calculation is shown in the figure below.

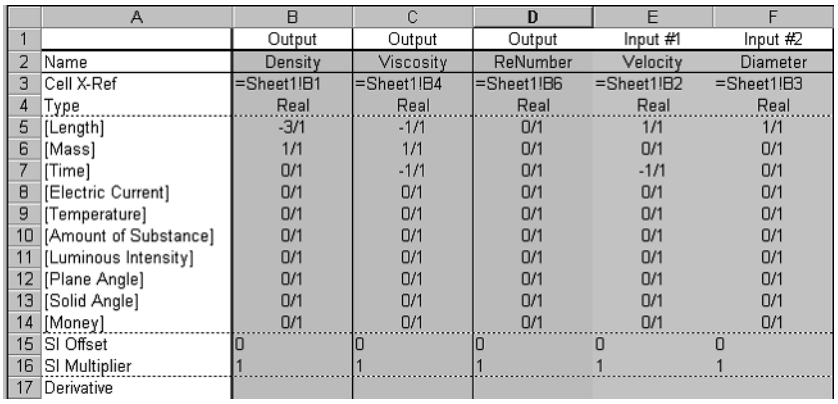
Figure 4.2. Worksheet containing the calculation of the Reynold's Number.



	A	B	C	D
1	Density	1000		
2	Velocity			
3	Diameter			
4	Viscosity	0.0001		
5				
6	Reynold's No	=B1*B2*B3/B4		
7				

The method-definition sheet shown in the figure below is easily created using the method definition macro. Simply run the macro (press ALT-F8), add a new method-definition worksheet (call it "methods"), add the three methods and add two inputs to the ReNumber method, selecting variable type "Real" for each. Return to the worksheet by pressing "OK" then "Exit" and complete the "gFO-methods" worksheet by adding the cross references and filling out the dimensions of the inputs and outputs (this last part is not necessary with the current version of gPROMS).

Figure 4.3. Macro-definition worksheet for the Reynold's Number example.



	A	B	C	D	E	F
1		Output	Output	Output	Input #1	Input #2
2	Name	Density	Viscosity	ReNumber	Velocity	Diameter
3	Cell X-Ref	=Sheet1!B1	=Sheet1!B4	=Sheet1!B6	=Sheet1!B2	=Sheet1!B3
4	Type	Real	Real	Real	Real	Real
5	[Length]	-3/1	-1/1	0/1	1/1	1/1
6	[Mass]	1/1	1/1	0/1	0/1	0/1
7	[Time]	0/1	-1/1	0/1	-1/1	0/1
8	[Electric Current]	0/1	0/1	0/1	0/1	0/1
9	[Temperature]	0/1	0/1	0/1	0/1	0/1
10	[Amount of Substance]	0/1	0/1	0/1	0/1	0/1
11	[Luminous Intensity]	0/1	0/1	0/1	0/1	0/1
12	[Plane Angle]	0/1	0/1	0/1	0/1	0/1
13	[Solid Angle]	0/1	0/1	0/1	0/1	0/1
14	[Money]	0/1	0/1	0/1	0/1	0/1
15	SI Offset	0	0	0	0	0
16	SI Multiplier	1	1	1	1	1
17	Derivative					

Cell ranges for arrays with more than two dimensions

Earlier, we looked at the specification of cell ranges for arrays of one or two dimensions. One-dimensional arrays can have cell ranges specified as either columns or rows. Variables and parameters defined as Array(*x*,*y*) correspond to cell ranges which are *y* cells wide and *x* cells deep. This is shown in the figure below:

(1,1)	(1,2)	(1,3)	...	(1,*y*)
(2,1)	(2,2)	(2,3)	...	(2,*y*)
...	...	...		...
(*x*,1)	(*x*,2)	(*x*,3)	...	(*x*,y)

Cell ranges must be specified as prescribed above because of the format in which gPROMS stores arrays and the way that the elements are sent to and read from Excel. Internally, gPROMS converts all n-dimensional arrays into arrays of one dimension. The elements of the n-dimensional array are mapped to the 1-d array by cycling through the indices; the right-most index is cycled fastest and the left-most index cycles through slowest. This is illustrated using a small 3-d array:

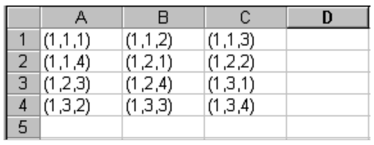
```
A1 AS ARRAY(2,3,4).
```

Internally, the elements of A1 would be stored in the order:

```
[ (1,1,1,), (1,1,2,), (1,1,3,), (1,1,4,), (1,2,1,), (1,2,2,),
  (1,2,3,), (1,2,4,), (1,3,1,), (1,3,2,), (1,3,3,), (1,3,4,),
  (2,1,1,), (2,1,2,), (2,1,3,), (2,1,4,), (2,2,1,), (2,2,2,),
  (2,2,3,), (2,2,4,), (2,3,1,), (2,3,2,), (2,3,3,), (2,3,4,) ]
```

When data is exchanged between gPROMS and Excel, the elements of arrays are read or sent in this order. In Excel, the elements are then read from or written to the cells starting with the top left cell of the cell range and moving left-to-right. If, for example, we were to specify the cell range A1:C8 for the above 3-dimensional array (obviously not a sensible choice), the first few rows would correspond to the following elements:

Figure 4.4. 3-D Array example



	A	B	C	D
1	(1,1,1)	(1,1,2)	(1,1,3)	
2	(1,1,4)	(1,2,1)	(1,2,2)	
3	(1,2,3)	(1,2,4)	(1,3,1)	
4	(1,3,2)	(1,3,3)	(1,3,4)	
5				

Clearly, the above cell-range specification is not much use, but since we now know how the elements are exchanged between Excel and gPROMS, we can specify a more useful cell range. For 3-dimensional arrays, there are two possibilities that would seem to be of most use: *z* cells wide and $x \times$ y cells deep or $y \times$ z cells wide and *x* cells deep, where the array is defined as Array(*x*,*y*,*z*). The former consists of *x* blocks of cells, stacked vertically, that are *z* cells wide and *y* cells deep; the latter consists of *y* blocks of cells, positioned horizontally, that are *z* cells wide and *x* cells deep. These two specifications are illustrated in the figures below.

Table 4.3. *z* by $x \times y$ cell range for an *x* by *y* by *z* array.

(1,1,1)	(1,1,2)	(1,1,3)	...	(1,1,*z*)
(1,2,1)	(1,2,2)	(1,2,3)	...	1,2,*z*)
...	...	...		...
(1,*y*,1)	(1,*y*,2)	(1,*y*,3)	...	(1,*y*,*z*)
(2,1,1)	(2,1,2)	(2,1,3)	...	(2,1,*z*)
(2,2,1)	(2,2,2)	(2,2,3)	...	(2,2,*z*)
...	...	...		...
(2,*y*,1)	(2,*y*,2)	(2,*y*,3)	...	(2,*y*,*z*)
		...		
(*x*,1,1)	(*x*,1,2)	(*x*,1,3)	...	(*x*,1,*z*)
(*x*,2,1)	(*x*,2,2)	(*x*,2,3)	...	(*x*,2,*z*)
...	...	...		...
(*x*,*y*,1)	(*x*,*y*,2)	(*x*,*y*,3)	...	(*x*,*y*,*z*)

Table 4.4. Block view

Block 1	Block 2	...	Block *y*

Table 4.5. Close-up view of block *j*

(1,*j*,1)	(1,*j*,2)	(1,*j*,3)	...	(1,*j*,*z*)
(2,*j*,1)	(2,*j*,2)	(2,*j*,3)	...	(2,*j*,*z*)
...	...	...		...
(*x*,*j*,1)	(*x*,*j*,2)	(*x*,*j*,3)	...	(*x*,*j*,*z*)

It is also possible to specify cell ranges for arrays of more dimensions. 4-dimensional arrays can be specified by repeating cell blocks for 3-dimensional arrays, and so on. However, in the majority of cases, it will not be necessary to specify cell ranges for more than 2-dimensional arrays.

Using the Microsoft Excel Foreign Process Interface

Setting up the gPROMS input file to use the Microsoft Excel FPI is very simple. All that is needed to enable the FPI is the following line in the Solutionparameters section:

```
FPI := "ExcelFP::filename"
```

where *filename* is the name of the Excel workbook to be used. Again, if the full path of the workbook is not specified, the file is assumed to be in the gPROMS export directory.

Once the FPI has been enabled using the above syntax, the following FPI tasks may be used in the Schedule section of a Process entity.

Pause

- gPROMS suspends execution of a process until the workbook is recalculated. Excel must be configured to recalculate only when the user requests it, which can be set in the "Tools/Options.../Calculation" menu.

Get

- gPROMS retrieves data from the Excel workbook and assigns them to a Variable (or Variables). This task behaves differently depending on the calculation mode set in the PFI worksheet (see later).

  Normal mode

  - In normal mode, the workbook is recalculated automatically and the data is retrieved.

  Manual mode

  - In manual mode, the execution of the process is suspended until the workbook is recalculated manually (as with the Pause task), at which point the required values are retrieved.

  Real-time mode

  - Real-time mode is used to synchronise the gPROMS simulation with real time. Every time a Get or Send task is called, gPROMS will pause until the real elapsed time is equal to the simulation time and then transmit or receive the data. Naturally, if the simulation is unable to keep up with real time (i.e. the model is too complex to solve in real time), this mode will behave similarly to Normal mode.

Send

- gPROMS sends the value(s) of one or more Variables to the Excel worksheet. If the "gTIME" tag has been defined, its value is automatically updated every time Send is called. Send is unaffected by the calculation mode.

Sendmathinfo

- Each time this task is called, a new worksheet is created and filled in with the details (current values of the variables, Jacobian elements, *etc*. of the current simulation problem.

The syntax for the above tasks is shown below.

```
PAUSE

  GET
    VariablePath := "ForeignVariableID" ;
  END

  SEND
    <SIGNALID "NoTime">
    "ForeignVariableID" := VariablePath ;
  END

  SENDMATHINFO
```

Pause and Sendmathinfo require no arguments. Get and Send specify the gPROMS variable pathname and the name of the Excel variable that are to be equated to each other. For the Get task, the gPROMS variable specified in *VariablePath* is assigned to the value of the Excel variable *ForeignVariableID* (this is defined in the FPI worksheet); for the Send task, the assignment works the other way around. Multiple assignments may be included in any one instance of the task. The Send task may also contain the optional argument:

```
SIGNALID "NoTime"
```

which suppresses the updating of "gTIME".

For array variables, the "*ForeignVariableID*" entry must also include the element range being sent (there is no shortcut for sending all elements, as there is in gPROMS). The syntax for this is:

```
"ForeignVariableID"(LowerCellLimit:UpperCellLimit < ,...>)
```

For example, if we wanted to send all elements of the variable A defined as an Array(2,4), the following syntax would be used:

```
   "A"(1:2,1:4) := A ;
```

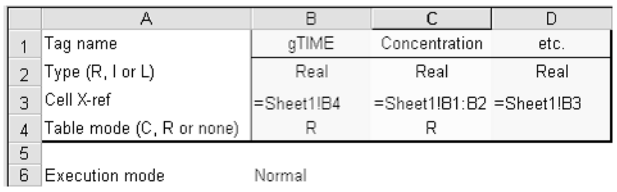
For a general description of the FPI communication tasks, see also: Elementary communication tasks for Foreign Processes.

A description of how the FPI is defined in the Excel workbook is now given.

Creating Microsoft Excel Foreign Processes

As with the FOI, the Excel FPI is defined using a worksheet with a special name. *ExcelFP* only uses one FPI definition sheet, called "gFPI". This worksheet defines each *tag* (the name that will be used in the gPROMS input file corresponding to *ForeignVariableID*), its cell cross-reference, the type variable (Real, Integer or Logical) and whether the data should be sent or received in tabular form. Finally, the "gFPI" worksheet contains a cell that defines the calculation mode (Normal, Real-time, or Manual). The format of the worksheet is shown in the figure below.

Figure 4.5. Format of the Foreign Process definition worksheet.



	A	B	C	D
1	Tag name	gTIME	Concentration	etc.
2	Type (R, I or L)	Real	Real	Real
3	Cell X-ref	=Sheet1!B4	=Sheet1!B1:B2	=Sheet1!B3
4	Table mode (C, R or none)	R	R	
5				
6	Execution mode	Normal		

As with the Foreign Object definition worksheets, the first column is ignored by *ExcelFP*. Each subsequent column defines a single tag. The name, type and cross-reference need no explanation (see also: Creating Microsoft Excel Foreign Objects for details on specifying types and cross-references). The table mode is used to specify how data sent to Excel may be tabulated or how tabulated data in Excel may be sent to gPROMS. If a table mode is specified, the *ExcelFPI* uses a cell range offset from the one specified in the FPI definition worksheet. Each time Get or Send is called on the tag, this offset increases so that either a table is created (Send) or read (Get). Specifying the table mode as R(owise) results in the cross-reference being increased by one row every time the tag is referenced in a Get or Send task (naturally, the original cell range must be either a scalar or a column vector and you cannot use both Send and Get tasks on the same table). The C(olumnwise) specification works similarly.

Finally, the execution mode is specified in cell B6. The execution mode affects how the Get task is handled, and may be set to Normal, Automatic or Manual. This was described in detail in: Using the Microsoft Excel Foreign Process Interface.

If a tag is defined with the name "gTIME", *ExcelFP* uses this to synchronise the execution of the gPROMS process with an external process. Whenever a Send task is called, *ExcelFP* automatically updates the gTIME tag with the current simulation time. If gTIME is being used in table mode, to generate a table of a variable with time as the abscissa, there are occasions when two Send tasks will be called in sequence without the simulation time increasing (e.g. at the beginning and end of a While loop). In these circumstances, two entries with the same time will be generated in the table. To avoid this, the

```
SIGNALID "NoTime"
```

option can be used to suppress one of the entries.

Using the FPI definition macro

As with *ExcelFO*, there is a macro available which greatly reduces the effort in creating the FPI definition worksheet. The macro is available in the gXLmacros.xls file and is accessed through the *Tools/Macro/Macros...* menu (or press ALT+F8). Selecting the MakeFPsheet and pressing "Run" will start the macro.

Using the FPI definition macro is very easy. Simply enter the details of the new tag in the appropriate boxes, then press the "Add" button to add a new tag. Existing tags may be removed by selecting its name and pressing the "Delete" button. Changes made in the session are updated on the worksheet by pressing "OK", whereas pressing "Cancel" discards all changes.

Troubleshooting

If gPROMS appears to have stopped working while using *ExcelFO* or *ExcelFP* then try the following:

- Make sure there are no dialog boxes open in Excel, and no cells are being edited. Then just select another cell in the worksheet. If *ExcelFO* or *ExcelFP* have trouble talking to Excel at any point, they often wait for either a selection change or a cell to be changed before continuing execution.
- Select all the cells in the worksheet, select the "format/cell..." menu, select the "protection" tab and make sure the "locked" property is unchecked. *ExcelFO* and *ExcelFP* use the locked property of cells to determine whether another instance is accessing those cells. This problem often occurs if a gPROMS run is stopped prematurely.

An example problem with Microsoft Excel FOI and FPI

Here we describe a more detailed example problem involving both the Microsoft Excel FOI and FPI. The process is a liquid-phase CSTR reactor involving the following reaction:

$A + B \rightleftharpoons C + D$

A general-purpose gPROMS model for an isothermal CSTR is shown in the listing below.

```
# MODEL LiquidPhaseCSTR

  PARAMETER
    NoComp             AS                          INTEGER
    NoReac             AS                          INTEGER
    ValveConstant      AS                          REAL
    CrossSectionalArea AS                          REAL
    Hp                 AS                          REAL
    Density            AS ARRAY(NoComp) OF         REAL
    ReactionConstant   AS ARRAY(NoReac) OF         REAL
    Order              AS ARRAY(NoComp,NoReac) OF INTEGER
    Nu                 AS ARRAY(NoComp,NoReac) OF INTEGER
    RxnData            AS                          FOREIGN_OBJECT "ExcelFO"

  VARIABLE
    Fin                AS                          MolarFlowrate
    Xin                AS ARRAY(NoComp) OF         MolarFraction
    Fout               AS                          MolarFlowrate
    X                  AS ARRAY(NoComp) OF         MolarFraction
    HoldUp             AS ARRAY(NoComp) OF         Moles
    C                  AS ARRAY(NoComp) OF         MolarConcentration
    TotalHoldup        AS                          Moles
    TotalVolume        AS                          Volume
    Height             AS                          Length
    Rate               AS ARRAY(NoReac) OF         NoType

  EQUATION
    FOR i := 1 TO NoComp DO
      $HoldUp(i) = Fin*Xin(i) - Fout*X(i) + TotalVolume*SIGMA(Nu(i,)*Rate) ;
    END

    FOR j := 1 TO NoReac DO
      Rate(j) = ReactionConstant(j) * PRODUCT(C^Order(,j)) ;
    END

    TotalVolume = SIGMA(Holdup/Density) ;

    TotalHoldup = SIGMA(HoldUp) ;

    Holdup = X * TotalHoldup ;

    Holdup = C * TotalVolume ;

    TotalVolume = CrossSectionalArea * Height ;

    IF Height > Hp THEN
      Fout = ValveConstant * (Height - Hp) ;
    ELSE
      Fout = 0 ;
    END
```

The reaction schemes and the rates of the reactions are specified through the parameters Nu, Order and ReactionConstant. The number of components and reactions are specified through the NoComp and NoReac parameters. A typical specification for the above reversible reaction is shown in the listing below.

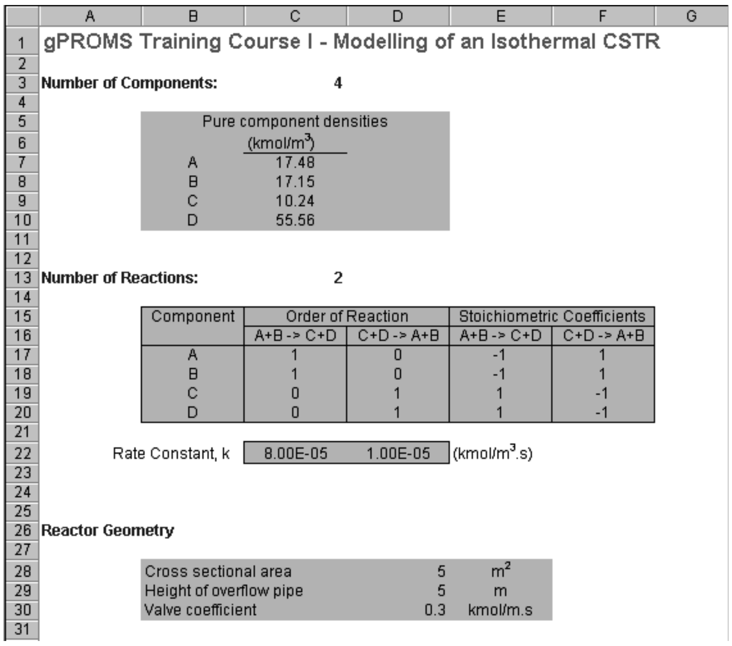
```
SET
    WITHIN R101 DO

      RxnData := "cstr.xls" ;

      NoComp  := 4 ;
      NoReac  := 2 ;
      Nu      := [ -1, +1,
                   -1, +1,
                   +1, -1,
                   +1, -1  ] ;
      Order   := [  1,  0,
                    1,  0,
                    0,  1,
                    0,  1  ] ;
      ReactionConstant   := [ 8E-5, 1E-5 ]                  ; # m3/kmol s
      Density            := [ 17.48, 17.15, 10.24, 55.56 ]  ; # kmol/m3
      CrossSectionalArea := 5                               ; # m2
      Hp                 := 5                               ; # m
      ValveConstant      := 0.3                             ;

    END
```

However, as this is quite cumbersome (and will be even more so with a large number of components and reactions), it would be more convenient to specify these using an Excel spreadsheet. The Foreign Object RxnData is instead used for this purpose. The listing below shows the equivalent parameter specification using the Foreign Object defined in the file cstr.xls.

```
SET
    WITHIN R101 DO

      RxnData             := "cstr.xls"                       ;

      NoComp              := RxnData.NumberOfComponents        ;
      NoReac              := RxnData.NumberOfReactions         ;
      Nu                  := RxnData.StoichiometricCoefficients ;
      Order               := RxnData.ReactionOrder             ;
      ReactionConstant    := RxnData.RateConstant              ;
      Density             := RxnData.Density                   ;
      CrossSectionalArea  := RxnData.CrossSectionalArea        ;
      ValveConstant       := RxnData.ValveCoefficient          ;
      Hp                  := RxnData.PipeHeight                ;

    END
```

The two worksheets that are used to generate the Foreign Object are shown in figures below. The first worksheet shows how the data for the parameters is entered. The second worksheet contains the method definitions for the Foreign Object (note the worksheet name begins with the prefix gFO-).

Figure 4.6. Parameter worksheet for the CSTR example.



gPROMS Training Course I - Modelling of an Isothermal CSTR

Number of Components: 4

	Pure component densities ($kmol/m^3$)
A	17.48
B	17.15
C	10.24
D	55.56

Number of Reactions: 2

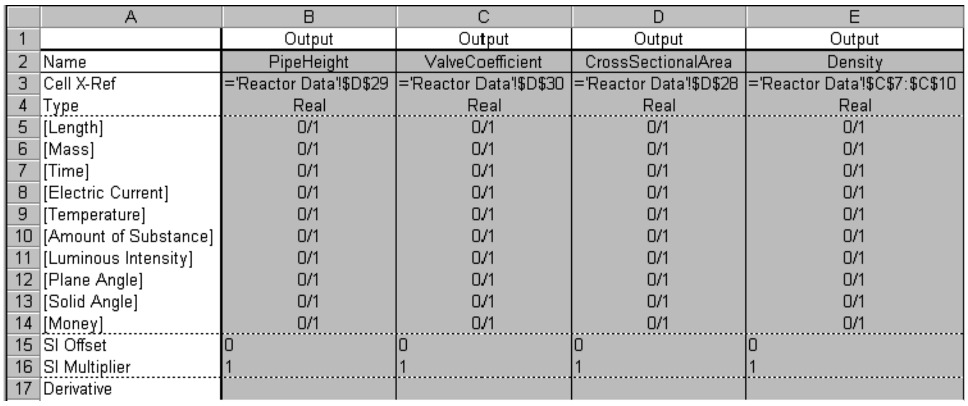
Component	Order of Reaction		Stoichiometric Coefficients	
	A+B -> C+D	C+D -> A+B	A+B -> C+D	C+D -> A+B
A	1	0	-1	1
B	1	0	-1	1
C	0	1	1	-1
D	0	1	1	-1

Rate Constant, k | 8.00E-05 | 1.00E-05 | ($kmol/m^3.s$)

Reactor Geometry

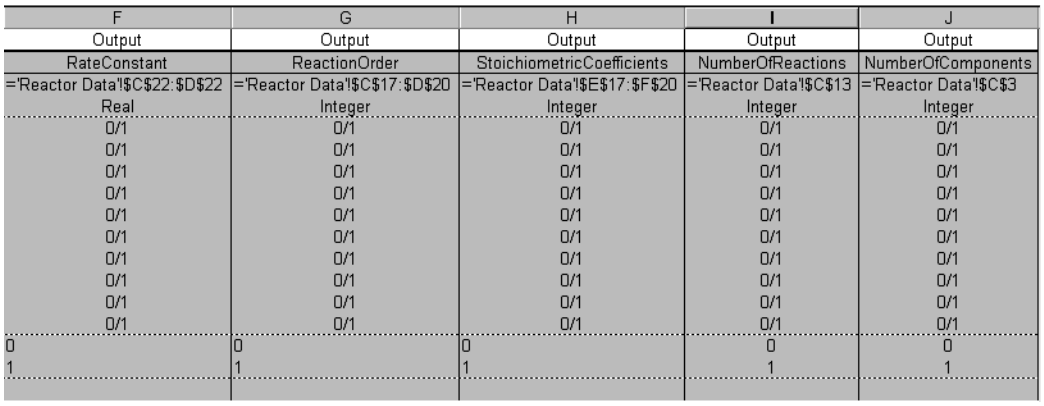
Cross sectional area	5	m^2
Height of overflow pipe	5	m
Valve coefficient	0.3	kmol/m.s

Figure 4.7. Method definition worksheet for the CSTR example - Columns A to E



	A	B	C	D	E
1		Output	Output	Output	Output
2	Name	PipeHeight	ValveCoefficient	CrossSectionalArea	Density
3	Cell X-Ref	='Reactor Data'!$D$29	='Reactor Data'!$D$30	='Reactor Data'!$D$28	='Reactor Data'!$C$7:$C$10
4	Type	Real	Real	Real	Real
5	[Length]	0/1	0/1	0/1	0/1
6	[Mass]	0/1	0/1	0/1	0/1
7	[Time]	0/1	0/1	0/1	0/1
8	[Electric Current]	0/1	0/1	0/1	0/1
9	[Temperature]	0/1	0/1	0/1	0/1
10	[Amount of Substance]	0/1	0/1	0/1	0/1
11	[Luminous Intensity]	0/1	0/1	0/1	0/1
12	[Plane Angle]	0/1	0/1	0/1	0/1
13	[Solid Angle]	0/1	0/1	0/1	0/1
14	[Money]	0/1	0/1	0/1	0/1
15	SI Offset	0	0	0	0
16	SI Multiplier	1	1	1	1
17	Derivative				

Figure 4.8. Method definition worksheet for the CSTR example - Columns F to J



F	G	H	I	J
Output	Output	Output	Output	Output
RateConstant	ReactionOrder	StoichiometricCoefficients	NumberOfReactions	NumberOfComponents
='Reactor Data'!$C$22:$D$22	='Reactor Data'!$C$17:$D$20	='Reactor Data'!$E$17:$F$20	='Reactor Data'!$C$13	='Reactor Data'!$C$3
Real	Integer	Integer	Integer	Integer
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0/1	0/1	0/1	0/1	0/1
0	0	0	0	0
1	1	1	1	1

Finally, the problem can be run in the gPROMS environment without an FPI using the typical Process entity shown in the listing below.

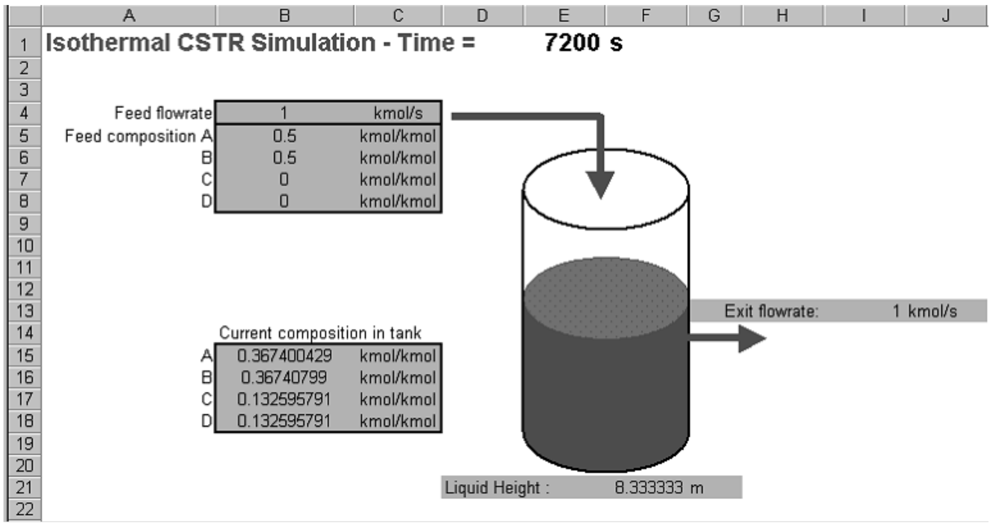
```
# PROCESS Sim2      # using EXCEL Foreign Object for data

  UNIT
    R101 AS LiquidPhaseCSTR

  SET
    ...

  ASSIGN
    WITHIN R101 DO
      Fin := 1                  ;
      Xin := [ 0.5, 0.5, 0, 0 ] ;
    END

  INITIAL
    WITHIN R101 DO
      X(2)        = 2 * X(1) ;
      X(3)        = 0        ;
      X(4)        = 0        ;
      TotalVolume = 10       ;
    END

  SCHEDULE
    CONTINUE FOR 7200
```

However, an alternative approach is to write a Microsoft Excel FPI to specify the inlet flowrate and composition and to retrieve the tank composition and outlet flowrate over time. This can be implemented by creating the worksheets shown below. The first worksheet simply contains a schematic of the process, along with the inlet conditions to be specified, the tank composition, the simulation time and the outlet flowrate. The second worksheet, called gFPI, contains the definitions of the tags that are used to link gPROMS variables with the cells in the Excel worksheet.

```
# PROCESS Sim3    # using EXCEL Foreign Object for data
                  # using EXCEL Foreign Process for real-time interaction

  UNIT
    R101 AS LiquidPhaseCSTR

  SET
    ...
```

```
ASSIGN
  WITHIN R101 DO
    Fin := 1                  ;
    Xin := [ 0.5, 0.5, 0, 0 ] ;
  END

INITIAL
  WITHIN R101 DO
    X(2)        = 2 * X(1) ;
    X(3)        = 0        ;
    X(4)        = 0        ;
    TotalVolume = 10       ;
  END

SOLUTIONPARAMETERS
  FPI := "ExcelFP::cstr.xls" ;

SCHEDULE
  WHILE TIME < 7200 DO
    SEQUENCE
      CONTINUE FOR 1
      PARALLEL
        GET
          R101.Fin := "FeedFlowrate"           ;
          R101.Xin := "FeedMoleFractions"(1:4) ;
        END
        SEND
          "TankMoleFractions"(1:4) := R101.X      ;
          "LiquidHeight"           := R101.Height ;
          "ExitFlowrate"           := R101.Fout   ;
        END
      END
    END
  END
```

Figure 4.9. User-interface worksheet for the CSTR example.



Isothermal CSTR Simulation - Time = 7200 s

Feed flowrate	1	kmol/s
Feed composition A	0.5	kmol/kmol
B	0.5	kmol/kmol
C	0	kmol/kmol
D	0	kmol/kmol

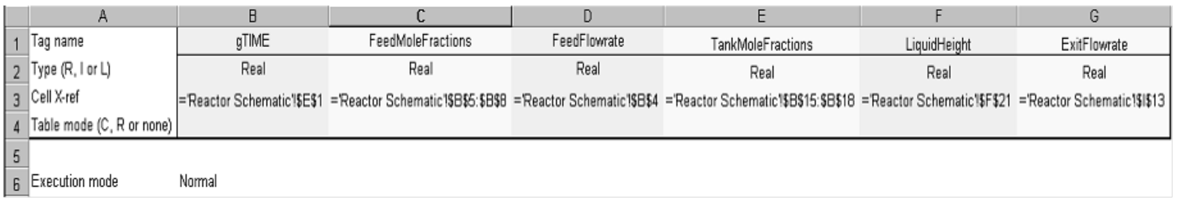
Exit flowrate: 1 kmol/s

Current composition in tank

A	0.367400429	kmol/kmol
B	0.36740799	kmol/kmol
C	0.132595791	kmol/kmol
D	0.132595791	kmol/kmol

Liquid Height : 8.333333 m

Figure 4.10. FPI definition worksheet for the CSTR example.



	A	B	C	D	E	F	G
1	Tag name	gTIME	FeedMoleFractions	FeedFlowrate	TankMoleFractions	LiquidHeight	ExitFlowrate
2	Type (R, I or L)	Real	Real	Real	Real	Real	Real
3	Cell X-ref	='Reactor Schematic'!$E$1	='Reactor Schematic'!$B$5:$B$8	='Reactor Schematic'!$B$4	='Reactor Schematic'!$B$15:$B$18	='Reactor Schematic'!$F$21	='Reactor Schematic'!$I$13
4	Table mode (C, R or none)						
5							
6	Execution mode	Normal					