

---

**AIMMS Function Reference - GMP Solver Session Procedures and Functions**

This file contains only one chapter of the book. For a free download of the complete book in pdf format, please visit [www.aimms.com](http://www.aimms.com)

# GMP::SolverSession Procedures and Functions

AIMMS supports the following procedures and functions for creating and managing solver sessions associated with a generated mathematical program instance:

- GMP::SolverSession::AddLinearization
- GMP::SolverSession::AsynchronousExecute
- GMP::SolverSession::CreateProgressCategory
- GMP::SolverSession::Execute
- GMP::SolverSession::ExecutionStatus
- GMP::SolverSession::GenerateBinaryEliminationRow
- GMP::SolverSession::GenerateBranchLowerBound
- GMP::SolverSession::GenerateBranchRow
- GMP::SolverSession::GenerateBranchUpperBound
- GMP::SolverSession::GenerateCut
- GMP::SolverSession::GetCallbackInterruptStatus
- GMP::SolverSession::GetCPUSecondsUsed
- GMP::SolverSession::GetInstance
- GMP::SolverSession::GetIterationsUsed
- GMP::SolverSession::GetLinearObjective
- GMP::SolverSession::GetMemoryUsed
- GMP::SolverSession::GetNodeNumber
- GMP::SolverSession::GetNodeObjective
- GMP::SolverSession::GetNodesLeft
- GMP::SolverSession::GetNodesUsed
- GMP::SolverSession::GetNumberOfBranchNodes
- GMP::SolverSession::GetObjective
- GMP::SolverSession::GetOptionValue
- GMP::SolverSession::GetProgramStatus
- GMP::SolverSession::GetSolver
- GMP::SolverSession::GetSolverStatus
- GMP::SolverSession::Interrupt
- GMP::SolverSession::RejectIncumbent
- GMP::SolverSession::SetObjective
- GMP::SolverSession::SetOptionValue
- GMP::SolverSession::Transfer

- `GMP::SolverSession::WaitForCompletion`
- `GMP::SolverSession::WaitForSingleCompletion`

---

## GMP::SolverSession::AddLinearization

The procedure `GMP::SolverSession::AddLinearization` adds a linearization row to a solver session with respect to a solution (column level values and row marginals) of a generated mathematical program for each row in a set of nonlinear constraints of that generated mathematical program.

```
GMP::SolverSession::AddLinearization(
    solverSession,    ! (input) a solver session
    GMP,              ! (input) a generated mathematical program
    solution,         ! (input) a solution
    constraintSet,    ! (input) a set of nonlinear constraints
    [jacTol],         ! (optional) the Jacobian tolerance
    [local],          ! (optional, default 0) a scalar value
    [purgeable]       ! (optional, default 0) a scalar binary expression
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*GMP*

An element in `AllGeneratedMathematicalPrograms`.

*solution*

An integer scalar reference to a solution in the solution repository of *GMP*.

*constraintSet*

A subset of `AllNonLinearConstraints`.

*jacTol*

The Jacobian tolerance; if the Jacobian value (in absolute sense) of a column in a nonlinear row is smaller than this value, the column will not be added to the linearization of that row. The default is  $1e-5$ .

*local*

A scalar binary value to indicate whether the linearization is valid for the local problem (i.e. the problem corresponding to the current node in the solution process and all its descendant nodes) only (value 1) or for the global problem (value 0).

*purgeable*

A scalar binary value to indicate whether the solver is allowed to purge the cut if it deems it ineffective. If the value is 1, then it is allowed.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

**Remarks:**

- This procedure fails if one of the constraints is ranged.
- This procedure can only be called from within a `CallbackAddCut` or `CallbackAddLazyConstraint` callback procedure.
- A `CallbackAddCut` callback procedure will only be called when solving mixed integer programs with CPLEX 8.0 or higher, or with XPRESS 17 or higher. In case of XPRESS the cuts are always local even if argument *local* has value 0.
- A `CallbackAddLazyConstraint` callback procedure will only be called when solving mixed integer programs with CPLEX 12.3 or higher.
- Argument *purgeable* can only be used with CPLEX 11.2 or higher. If the cut is local then the cut will not be purgeable even if argument *purgeable* has value 1.
- This procedure will fail if *GMP* contains a column that is not part of the generated mathematical program corresponding to *solverSession*. A column that is part of *GMP* but not of the generated mathematical program corresponding to *solverSession* will be ignored, i.e., no coefficient for that column will be added to the linearizations.
- The formula for the linearization of a scalar nonlinear inequality  $g(x, y) \leq b_j$  around the point  $(x, y) = (x^0, y^0)$  is as follows.

$$g(x^0, y^0) + \nabla g(x^0, y^0)^T \begin{bmatrix} x - x^0 \\ y - y^0 \end{bmatrix} \leq b_j$$

**See also:**

The routines `GMP::Linearization::Add`, `GMP::Instance::SetCallbackAddLazyConstraint` and `GMP::SolverSession::GenerateCut`.

---

## GMP::SolverSession::AsynchronousExecute

The procedure `GMP::SolverSession::AsynchronousExecute` invokes the solution algorithm to asynchronously solve a generated mathematical program by using a solver session.

```
GMP::SolverSession::AsynchronousExecute(
    solverSession ! (input) a solver session
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- The following solvers are thread-safe and can be used for solving multiple mathematical programs in parallel using the same solver: CPLEX, GUROBI, MOSEK, XA, CONOPT 3.14V or higher, and KNITRO 6.0 or higher.
- The following solvers are not thread-safe but the AIMMS-solver interface is thread safe and therefore they can be used in parallel with another solver: CONOPT 3.14Q or lower, IPOPT, KNITRO 5.2 or lower, and SNOPT. For example, SNOPT 7.1 can be used in parallel with IPOPT but it cannot be used in parallel with SNOPT 7.1.
- The procedure `GMP::SolverSession::AsynchronousExecute` cannot be used by the following solvers: AOA, BARON, CBC, LGO, and PATH.
- Calling `GMP::SolverSession::AsynchronousExecute` inside a callback procedure is not allowed.
- The procedures `GMP::SolverSession::WaitForCompletion` and `GMP::SolverSession::WaitForSingleCompletion` can be used to let AIMMS wait until one or more asynchronous executing solver sessions are finished.
- If you want to asynchronously execute two solver sessions belonging to the same GMP, you should create a copy of that GMP by using the function `GMP::Instance::Copy` and create the second solver session for that copy.
- The procedure `GMP::SolverSession::AsynchronousExecute` is not supported for solvers that use the Proxy Stub interface.
- Normal solve statements will be ignored during an asynchronous execution of a solver session.
- This procedure does not create a listing file.

**See also:**

The routines `GMP::Instance::Copy`, `GMP::SolverSession::Execute`, `GMP::SolverSession::ExecutionStatus`, `GMP::SolverSession::Interrupt`, `GMP::SolverSession::WaitForCompletion` and `GMP::SolverSession::WaitForSingleCompletion`.

---

## GMP::SolverSession::CreateProgressCategory

The function `GMP::SolverSession::CreateProgressCategory` creates a new progress category for a solver session. This progress category can be used to display solver (session) related information in the Progress Window.

There are three levels of progress categories for solver information. By default all solver progress will be displayed in the general AIMMS progress category for solver progress. If a progress category was created for the GMP with procedure `GMP::Instance::CreateProgressCategory`, then all solver progress related to that GMP will by default be displayed in the solver progress category of the GMP. For displaying solver session progress in a separated category the function `GMP::SolverSession::CreateProgressCategory` can be used.

```
GMP::SolverSession::CreateProgressCategory(
    solverSession,    ! (input) a solver session
    [Name],          ! (optional) a string expression
    [Size]           ! (optional) an integer expression
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*Name*

A string that holds the name of the progress category.

*Size*

The number of lines in the progress category. The default is 0 meaning that the size of the progress window will be automatically adjusted to the number of progress lines used by the solver.

### Return value:

The function returns an element in the set `AllProgressCategories`.

### Remarks:

- If the *Name* argument is not specified then the name of the solver session will be used to name the element in the set `AllProgressCategories`.
- The information displayed in the solver session progress window can be controlled by using the procedures `GMP::ProgressWindow::DisplayLine` and `GMP::ProgressWindow::FreezeLine`.
- A progress category created before for the solver session will be deleted.

### See also:

The routines `GMP::ProgressWindow::CreateProgressCategory`, `GMP::ProgressWindow::DeleteCategory`, `GMP::ProgressWindow::DisplayLine`, `GMP::ProgressWindow::FreezeLine` and `GMP::ProgressWindow::UnfreezeLine`.

---

**GMP::SolverSession::Execute**

The procedure `GMP::SolverSession::Execute` invokes the solution algorithm to solve the mathematical program for which it had been generated.

```
GMP::SolverSession::Execute(  
    solverSession  ! (input) a solver session  
)
```

**Arguments:**

*solverSession*  
An element in the set `AllSolverSessions`.

**Return value:**

The procedure returns 1 on success, or 0 otherwise.

**Remarks:**

This procedure does not create a listing file.

**See also:**

The routines `GMP::Instance::CreateSolverSession`, `GMP::Instance::Solve` and `GMP::SolverSession::AsynchronousExecute`.

---

## GMP::SolverSession::ExecutionStatus

The function `GMP::SolverSession::ExecutionStatus` returns the execution status of a solver session.

```
GMP::SolverSession::ExecutionStatus(  
    solverSession    ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

An element in the set `AllExecutionStatuses`. The set `AllExecutionStatuses` contains the following elements:

- `NotStarted`,
- `Pending`,
- `Running`,
- `Interrupted`,
- `Finished`.

### See also:

The routines `GMP::SolverSession::AsynchronousExecute`, `GMP::SolverSession::Interrupt`, `GMP::SolverSession::WaitForCompletion` and `GMP::SolverSession::WaitForSingleCompletion`.

---

## GMP::SolverSession::GenerateBinaryEliminationRow

The procedure `GMP::SolverSession::GenerateBinaryEliminationRow` adds a binary row to a solver session which will eliminate a binary solution.

```
GMP::SolverSession::GenerateBinaryEliminationRow(
    solverSession,    ! (input) a solver session
    solution,        ! (input) a solution
    branch           ! (input) a scalar value
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*solution*

An integer scalar reference to a solution.

*branch*

An integer scalar reference to a branch. Value should be either 1 or 2.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- This procedure will fail if the GMP corresponding to the solver session does not have model type MIP.
- This procedure can only be called from within a `CallbackBranch` or `CallbackAddCut` callback procedure. In the latter case the *branch* argument will be ignored.
- Every call to `GMP::SolverSession::GenerateBinaryEliminationRow` adds the following row:

$$\sum_{i \in S_{lo}} x_i - \sum_{i \in S_{up}} x_i \geq 1 - \sum_{i \in S_{up}} lev_i \quad (37.1)$$

where  $S_{lo}$  defines the set of binary columns whose level values equals 0 and  $S_{up}$  the set of binary columns whose level values equals 1.

### Examples:

The procedure `GMP::SolverSession::GenerateBinaryEliminationRow` can be used to enforce a MIP solver to branch a node that would have been fathomed otherwise. We can achieve this by installing a branching callback using procedure `GMP::Instance::SetCallbackBranch` and adding the following code to the callback procedure:

```
! Get LP solution at the current node.
GMP::Solution::RetrieveFromSolverSession(ThisSession,1);
! Get the number of nodes that the MIP solver wants to create from the
! current branch.
NrBranches := GMP::SolverSession::GetNumberOfBranchNodes(ThisSession);
if ( NrBranches = 0 ) then
    ! The LP solution at the current node appears to be integer feasible.
    ! We enforce the MIP solver to branch the current node by creating a
    ! branch containing one constraint that cuts off this LP solution.
    GMP::SolverSession::GenerateBinaryEliminationRow(ThisSession,1,1);
endif;
```

Here 'ThisSession' is an input argument of the callback procedure and a scalar element parameter into the set AllSolverSessions.

**See also:**

The routines `GMP::Instance::AddIntegerEliminationRows`, `GMP::Instance::SetCallbackAddCut`, `GMP::Instance::SetCallbackBranch`, and `GMP::SolverSession::GetNumberOfBranchNodes`.

---

## GMP::SolverSession::GenerateBranchLowerBound

The procedure `GMP::SolverSession::GenerateBranchLowerBound` specifies the lower bound change of a column in a branch to be taken from the current node during MIP branch & cut.

```
GMP::SolverSession::GenerateBranchLowerBound(
    solverSession,    ! (input) a solver session
    column,          ! (input) a scalar reference
    bound,           ! (input) a numerical expression
    branch           ! (input) a branch number
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*column*

A scalar reference to an existing column in the model.

*bound*

The value assigned to the lower bound change of the column in the branch.

*branch*

An integer scalar reference to the branch number. It should be equal to 1 or 2.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- A branch can be specified by adding multiple bound changes and rows (with `GMP::SolverSession::GenerateBranchRow`) to the node problem.
- This procedure can only be called from within a `CallbackBranch` callback procedure.
- A `CallbackBranch` callback procedure will only be called when solving mixed integer programs with CPLEX 9.1 or higher.

### See also:

The procedures `GMP::Instance::SetCallbackBranch`, `GMP::SolverSession::GenerateBranchUpperBound` and `GMP::SolverSession::GenerateBranchRow`.

---

## GMP::SolverSession::GenerateBranchRow

The procedure `GMP::SolverSession::GenerateBranchRow` adds a row to a branch to be taken from the current node during MIP branch & cut.

```
GMP::SolverSession::GenerateBranchRow(
    solverSession,    ! (input) a solver session
    row,              ! (input) a scalar reference
    branch            ! (input) a branch number
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*row*

A scalar reference to an existing row in the model.

*branch*

An integer scalar reference to the branch number. It should be equal to 1 or 2.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- A branch can be specified by adding multiple rows and bound changes (with `GMP::SolverSession::GenerateBranchLowerBound` and `GMP::SolverSession::GenerateBranchUpperBound`) to the node problem.
- This procedure can only be called from within a `CallbackBranch` callback procedure.
- A `CallbackBranch` callback procedure will only be called when solving mixed integer programs with CPLEX 9.1 or higher.

### See also:

The procedures `GMP::Instance::SetCallbackBranch`, `GMP::SolverSession::GenerateBranchLowerBound` and `GMP::SolverSession::GenerateBranchUpperBound`.

---

## GMP::SolverSession::GenerateBranchUpperBound

The procedure `GMP::SolverSession::GenerateBranchUpperBound` specifies the upper bound change of a column in a branch to be taken from the current node during MIP branch & cut.

```
GMP::SolverSession::GenerateBranchUpperBound(
    solverSession,    ! (input) a solver session
    column,          ! (input) a scalar reference
    bound,           ! (input) a numerical expression
    branch           ! (input) a branch number
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*column*

A scalar reference to an existing column in the model.

*bound*

The value assigned to the upper bound change of the column in the branch.

*branch*

An integer scalar reference to the branch number. It should be equal to 1 or 2.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- A branch can be specified by adding multiple bound changes and rows (with `GMP::SolverSession::GenerateBranchRow`) to the node problem.
- This procedure can only be called from within a `CallbackBranch` callback procedure.
- A `CallbackBranch` callback procedure will only be called when solving mixed integer programs with CPLEX 9.1 or higher.

### See also:

The procedures `GMP::Instance::SetCallbackBranch`, `GMP::SolverSession::GenerateBranchLowerBound` and `GMP::SolverSession::GenerateBranchRow`.

---

## GMP::SolverSession::GenerateCut

The procedure `GMP::SolverSession::GenerateCut` adds a cut to the LP subproblem of the current node during MIP branch & cut. It can also be used to add a lazy constraint inside a callback for adding lazy constraints.

```
GMP::SolverSession::GenerateCut(
    solverSession,    ! (input) a solver session
    row,              ! (input) a scalar reference
    [local],          ! (optional, default 0) a scalar binary expression
    [purgeable]      ! (optional, default 0) a scalar binary expression
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*row*

A scalar reference to an existing row in the model.

*local*

A scalar binary value to indicate whether the cut is valid for the local problem (i.e. the problem corresponding to the current node in the solution process and all its descendant nodes) only (value 1) or for the global problem (value 0).

*purgeable*

A scalar binary value to indicate whether the solver is allowed to purge the cut if it deems it ineffective. If the value is 1, then it is allowed.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- This procedure can only be called from within a `CallbackAddCut` or `CallbackAddLazyConstraint` callback procedure.
- A `CallbackAddCut` callback procedure will only be called when solving mixed integer programs with CPLEX 8.0 or higher, with GUROBI 2.0 or higher, or with XPRESS 17 or higher. In case of GUROBI and XPRESS the cuts are always local even if argument *local* has value 0.
- A `CallbackAddLazyConstraint` callback procedure will only be called when solving mixed integer programs with CPLEX 12.3 or higher
- Argument *purgeable* can only be used with CPLEX 11.2 or higher. If the cut is local then the cut will not be purgeable even if argument *purgeable* has value 1.
- This procedure can also be used for MIQP and MIQCP problems.

**See also:**

The procedures `GMP::Instance::SetCallbackAddCut` and `GMP::Instance::SetCallbackAddLazyConstraint`. See Section 21.2 of the Language Reference for more details on how to install a callback procedure to add cuts.

---

## GMP::SolverSession::GetCallbackInterruptStatus

The function `GMP::SolverSession::GetCallbackInterruptStatus` returns the type of the last callback function that had been called during a specific solver session.

```
GMP::SolverSession::GetCallbackInterruptStatus(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

An element in the set `AllSolverInterrupts`.

### Remarks:

When the solver session has not yet been executed, the empty element will be returned.

### See also:

The procedures `GMP::SolverSession::Execute`, `GMP::Instance::SetCallbackAddCut`, `GMP::Instance::SetCallbackAddLazyConstraint`, `GMP::Instance::SetCallbackIncumbent`, `GMP::Instance::SetCallbackIterations`, `GMP::Instance::SetCallbackHeuristic`, `GMP::Instance::SetCallbackNewIncumbent` and `GMP::Instance::SetCallbackStatusChange`.

---

**GMP::SolverSession::GetCPUSecondsUsed**

The function `GMP::SolverSession::GetCPUSecondsUsed` returns the time (in 1/100th seconds) needed to execute a solver session.

```
GMP::SolverSession::GetCPUSecondsUsed(  
    solverSession  ! (input) a solver session  
)
```

**Arguments:**

*solverSession*  
An element in the set `AllSolverSessions`.

**Return value:**

The number of 1/100th seconds used to execute a solver session.

**See also:**

The routines `GMP::Instance::SetCPUSecondsLimit`, `GMP::SolverSession::Execute`, `GMP::SolverSession::GetIterationsUsed` and `GMP::SolverSession::GetMemoryUsed`.

---

**GMP::SolverSession::GetInstance**

The function `GMP::SolverSession::GetInstance` returns the generated mathematical program that was used to create a solver session.

```
GMP::SolverSession::GetInstance(  
    solverSession  ! (input) a solver session  
)
```

**Arguments:**

*solverSession*  
An element in the set `AllSolverSessions`.

**Return value:**

An element in the set `AllGeneratedMathematicalPrograms`.

**See also:**

The routines `GMP::Instance::Generate` and `GMP::Instance::CreateSolverSession`.

---

## GMP::SolverSession::GetIterationsUsed

The function `GMP::SolverSession::GetIterationsUsed` returns the number of iterations used by a solver session.

```
GMP::SolverSession::GetIterationsUsed(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The number of iterations used by the solver session.

### See also:

The routines `GMP::SolverSession::Execute`, `GMP::Instance::SetIterationLimit`, `GMP::SolverSession::GetCPUSecondsUsed` and `GMP::SolverSession::GetMemoryUsed`.

---

## GMP::SolverSession::GetLinearObjective

The function `GMP::SolverSession::GetLinearObjective` returns the best known bound for a solver session.

```
GMP::SolverSession::GetLinearObjective(  
    solverSession    ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

In case of success, the best known bound. Otherwise it returns `UNDF`.

### Remarks:

This function has only meaning for solver sessions with a corresponding generated mathematical program that has model type `MIP`, `MIQP` or `MIQCP`.

### See also:

The routines `GMP::SolverSession::Execute`, `GMP::SolverSession::GetObjectiv`, `GMP::SolverSession::GetIterationsUsed`, `GMP::SolverSession::GetCPUSecondsUsed` and `GMP::SolverSession::GetMemoryUsed`.

---

## GMP::SolverSession::GetMemoryUsed

The function `GMP::SolverSession::GetMemoryUsed` returns the amount of memory used by the solver session.

```
GMP::SolverSession::GetMemoryUsed(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The amount of megabytes used to execute a solver session.

### See also:

The routines `GMP::Instance::SetMemoryLimit`, `GMP::SolverSession::Execute`, `GMP::SolverSession::GetIterationsUsed` and `GMP::SolverSession::GetCPUSecondsUsed`.

---

## GMP::SolverSession::GetNodeNumber

The function `GMP::SolverSession::GetNodeNumber` returns the number of the current node during MIP optimization from within a node callback.

```
GMP::SolverSession::GetNodeNumber(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The number of the node for which the callback is called. It returns -1 if this function is not called inside a solver callback, or if it is not supported by the solver.

### Remarks:

- This function has only meaning for solver sessions belonging to a GMP with type MIP, MIQP or MIQCP.
- This function can only be used inside a **branch, cuts, heuristic** or **incumbent** callback.
- This function is only supported by CPLEX 8.0 or higher and by XPRESS 17 or higher.
- The root node in a branch-and-bound tree gets number 0.

### See also:

The routines `GMP::Instance::SetCallbackAddCut`, `GMP::Instance::SetCallbackBranch`, `GMP::Instance::SetCallbackHeuristic`, `GMP::Instance::SetCallbackIncumbent` and `GMP::SolverSession::GetNodesUsed`.

---

## GMP::SolverSession::GetNodeObjective

The function `GMP::SolverSession::GetNodeObjective` returns the objective value for the subproblem at the current node during MIP optimization from within a node callback.

```
GMP::SolverSession::GetNodeObjective(
    solverSession    ! (input) a solver session
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

In case of success, the objective value at the current node. Otherwise it returns UNDF.

### Remarks:

- This function has only meaning for solver sessions belonging to a GMP with type MIP, MIQP or MIQCP.
- This function can only be used inside a **branch**, **cuts**, **heuristic**, **incumbent** or **lazy constraint** callback.
- The procedure `GMP::Solution::RetrieveFromSolverSession` can be used to retrieve the node solution inside a **branch**, **cuts**, **heuristic** or **lazy constraint** callback.
- This function is only supported by CPLEX 8.0 or higher and by XPRESS 17 or higher.

### See also:

The routines `GMP::Instance::SetCallbackAddCut`, `GMP::Instance::SetCallbackAddLazyConstraint`, `GMP::Instance::SetCallbackBranch`, `GMP::Instance::SetCallbackHeuristic`, `GMP::Instance::SetCallbackIncumbent`, `GMP::Solution::RetrieveFromSolverSession` and `GMP::SolverSession::GetNodeNumber`.

---

## GMP::SolverSession::GetNodesLeft

The function `GMP::SolverSession::GetNodesLeft` returns the number of unexplored nodes left in the branch-and-bound tree for a solver session.

```
GMP::SolverSession::GetNodesLeft(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The number of unexplored nodes left in the branch-and-bound tree.

### Remarks:

- This function has only meaning for solver sessions belonging to a GMP with type MIP, MIQP or MIQCP.
- This function can be used inside a **branch, cuts, heuristic** or **incumbent** callback.
- This function is supported by CPLEX 8.0, GUROBI 2.0, MOSEK 6.0, XPRESS 17, and all higher versions.

### See also:

The routines `GMP::Instance::SetCallbackAddCut`, `GMP::Instance::SetCallbackBranch`, `GMP::Instance::SetCallbackHeuristic`, `GMP::Instance::SetCallbackIncumbent`, `GMP::SolverSession::GetNodeNumber` and `GMP::SolverSession::GetNodesUsed`.

---

## GMP::SolverSession::GetNodesUsed

The function `GMP::SolverSession::GetNodesUsed` returns the number of nodes that are processed by a solver session.

```
GMP::SolverSession::GetNodesUsed(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The number of nodes that are processed by the solver session.

### Remarks:

- This function has only meaning for solver sessions belonging to a GMP with type MIP, MIQP or MIQCP.
- This function can be used inside a **branch**, **cuts**, **heuristic** or **incumbent** callback.

### See also:

The routines `GMP::Instance::SetCallbackAddCut`, `GMP::Instance::SetCallbackBranch`, `GMP::Instance::SetCallbackHeuristic`, `GMP::Instance::SetCallbackIncumbent`, `GMP::SolverSession::GetNodeNumber` and `GMP::SolverSession::GetNodesLeft`.

---

## GMP::SolverSession::GetNumberOfBranchNodes

The function `GMP::SolverSession::GetNumberOfBranchNodes` returns the number of nodes that the solver will create from the current branch.

```
GMP::SolverSession::GetNumberOfBranchNodes(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The number of nodes that the solver will create from the current branch.

### Remarks:

- If the value returned equals 0, the node will be fathomed unless user-specified branches are made. That is, no child nodes are created and the node itself is discarded.
- This function has only meaning for solver sessions belonging to a GMP with type MIP, MIQP or MIQCP.
- This function can be used inside a **branch** callback.

### See also:

The routines `GMP::Instance::SetCallbackBranch`.

---

## GMP::SolverSession::GetObjective

The function `GMP::SolverSession::GetObjective` returns the objective function value associated with a solver session.

```
GMP::SolverSession::GetObjective(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The objective function value associated with a solver session.

### See also:

The routines `GMP::SolverSession::Execute`, `GMP::SolverSession::GetLinearObjective`, `GMP::SolverSession::GetIterationsUsed`, `GMP::SolverSession::GetCPUSecondsUsed`, `GMP::SolverSession::GetMemoryUsed` and `GMP::SolverSession::SetObjective`.

---

## GMP::SolverSession::GetOptionValue

The function `GMP::SolverSession::GetOptionValue` returns the value of a solver specific option for a solver session.

```
GMP::SolverSession::GetOptionValue(  
    solverSession,    ! (input) a solver session  
    OptionName       ! (input) a scalar string expression  
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*OptionName*

A string expression holding the name of the option.

### Return value:

In case of success, the function returns the current option value. Otherwise it returns UNDF.

### Remarks:

Options for which strings are displayed in the AIMMS **Options** dialog box, are also represented by numerical (integer) values. To obtain the corresponding option keywords, you can use the functions `OptionGetString` and `OptionGetKeywords`.

### See also:

The routines `GMP::Instance::GetOptionValue`, `GMP::Instance::SetOptionValue`, `GMP::SolverSession::SetOptionValue`, `OptionGetString` and `OptionGetKeywords`.

---

**GMP::SolverSession::GetProgramStatus**

The function `GMP::SolverSession::GetProgramStatus` returns the program status of the last execution of a solver session.

```
GMP::SolverSession::GetProgramStatus(  
    solverSession  ! (input) a solver session  
)
```

**Arguments:**

*solverSession*  
An element in the set `AllSolverSessions`.

**Return value:**

An element in the set `AllSolutionStates`.

**See also:**

The routines `GMP::SolverSession::Execute` and `GMP::SolverSession::GetSolverStatus`.

---

## GMP::SolverSession::GetSolver

The function `GMP::SolverSession::GetSolver` returns the solver belonging to a solver session.

```
GMP::SolverSession::GetSolver(  
    solverSession    ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The solver belonging to a solver session as an element of `AllSolvers`.

### Remarks:

Which solver is assigned to the solver session is determined by the routines `GMP::Instance::CreateSolverSession` and `GMP::Instance::SetSolver`. Note that if the *Solver* argument of `GMP::Instance::CreateSolverSession` is used then it overrules `GMP::Instance::SetSolver`.

### See also:

The routines `GMP::Instance::CreateSolverSession` and `GMP::Instance::SetSolver`.

---

**GMP::SolverSession::GetSolverStatus**

The function `GMP::SolverSession::GetSolverStatus` returns the solver status of the last execution of a solver session.

```
GMP::SolverSession::GetSolverStatus(  
    solverSession  ! (input) a solver session  
)
```

**Arguments:**

*solverSession*  
An element in the set `AllSolverSessions`.

**Return value:**

An element in the set `AllSolutionStates`.

**See also:**

The routines `GMP::SolverSession::Execute` and `GMP::SolverSession::GetProgramStatus`.

---

## GMP::SolverSession::Interrupt

The procedure `GMP::SolverSession::Interrupt` interrupts a solver session that is (asynchronous) executing.

```
GMP::SolverSession::Interrupt(  
    solverSession,    ! (input) a solver session  
    [timeout]        ! (optional) timeout interval  
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*timeout*

A scalar value indicating the time-out interval (in seconds). The default value is 600.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- This interrupt procedure will wait until the solver session is successfully interrupted or the time-out interval elapses.
- This procedure can also be called for a solver session that is not asynchronous executing. In that case the *timeout* argument will be ignored.

### See also:

The routines `GMP::SolverSession::AsynchronousExecute`, `GMP::SolverSession::ExecutionStatus`, `GMP::SolverSession::Interrupt`, `GMP::SolverSession::WaitForCompletion` and `GMP::SolverSession::WaitForSingleCompletion`.

---

## GMP::SolverSession::RejectIncumbent

The procedure `GMP::SolverSession::RejectIncumbent` rejects the integer solution found by a solver session during the solution process of a MIP model.

```
GMP::SolverSession::RejectIncumbent(  
    solverSession  ! (input) a solver session  
)
```

### Arguments:

*solverSession*  
An element in the set `AllSolverSessions`.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

- This procedure can only be called from within a `CallbackIncumbent` callback procedure.
- A `CallbackIncumbent` callback procedure will only be called when solving mixed integer programs with CPLEX 8.0 or higher, or with XPRESS 19 or higher.

### See also:

The procedure `GMP::Instance::SetCallbackIncumbent`. See Section 21.2 of the Language Reference for more details on how to install an incumbent callback procedure.

---

## GMP::SolverSession::SetObjective

The procedure `GMP::SolverSession::SetObjective` sets the objective value for the solution belonging to a solver session.

```
GMP::SolverSession::SetObjective(  
    solverSession,    ! (input) a solver session  
    Value             ! (input) a scalar numeric expression  
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*Value*

A scalar numeric expression representing the new value to be assigned as the objective value.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### See also:

The routine `GMP::SolverSession::Execute` and `GMP::SolverSession::GetObjective`.

---

## GMP::SolverSession::SetOptionValue

The procedure `GMP::SolverSession::SetOptionValue` sets the value of a solver specific option for a solver session. To a solver session corresponds to one unique solver, and the option will only be set for that solver.

```
GMP::SolverSession::SetOptionValue(
    solverSession,    ! (input) a solver session
    OptionName,      ! (input) a scalar string expression
    Value            ! (input) a scalar numeric expression
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*OptionName*

A string expression holding the name of the option.

*Value*

A scalar numeric expression representing the new value to be assigned to the option.

### Return value:

The procedure returns 1 if the option exists and the value can be assigned to the option, or 0 otherwise.

### Remarks:

- The option value of a solver specific option can also be set in other ways. The value of an option belonging to a solver session is determined by:
  - the procedure `GMP::SolverSession::SetOptionValue` if it is called for the solver session, else
  - the procedure `GMP::Instance::SetOptionValue` if it is called for the generated mathematical program corresponding to the solver session, else
  - the value used in the `OPTION` statement if that statement is used (see also Section 8.5 of the Language Reference), else
  - the option value in the option tree.
- Options for which strings are displayed in the AIMMS **Options** dialog box, are also represented by numerical (integer) values. To obtain the corresponding option keywords, you can use the functions `OptionGetString` and `OptionGetKeywords`.

### See also:

The routines `GMP::Instance::GetOptionValue`, `GMP::Instance::SetOptionValue`, `GMP::SolverSession::GetOptionValue`, `OptionGetString` and `OptionGetKeywords`.

---

## GMP::SolverSession::Transfer

The procedure `GMP::SolverSession::Transfer` can be used to transfer a solver session from its current GMP to another similar GMP. Both GMPs should be created from the same symbolic math program.

Currently this procedure is only supported for stochastic Benders decomposition.

```
GMP::SolverSession::Transfer(
    solverSession, ! (input) a solver session
    GMP           ! (input) a generated mathematical program
)
```

### Arguments:

*solverSession*

An element in the set `AllSolverSessions`.

*GMP*

An element in the set `AllGeneratedMathematicalPrograms`.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

If each GMP has its own solver session then more memory is required which might not be available for large models or if many GMPs are used. To save memory this procedure can be used since it allows similar GMPs to share one solver session. After transferring a solver session to a *GMP*, only the differences between the old and new GMP will be passed as updates to the solver.

### See also:

The routines `GMP::Instance::CreateSolverSession`, `GMP::Instance::GenerateStochasticProgram` and `GMP::Stochastic::BendersFindReference`.

---

## GMP::SolverSession::WaitForCompletion

The procedure `GMP::SolverSession::WaitForCompletion` has a set of objects as its input. The set of objects may contain solver sessions that are asynchronous executing and events. This procedure lets AIMMS wait until all the solver sessions have completed their asynchronous execution and all the events get activated.

```
GMP::SolverSession::WaitForCompletion(  
    solSesSet      ! (input) a set of objects  
)
```

### Arguments:

*solSesSet*  
A subset of `AllSolverSessionCompletionObjects`.

### Return value:

The procedure returns 1 on success, or 0 otherwise.

### Remarks:

This procedure ignores solver sessions that are not asynchronous executing but using the procedure `GMP::SolverSession::Execute`.

### See also:

The routines `GMP::Event::Create`, `GMP::Event::Set`, `GMP::SolverSession::AsynchronousExecute`, `GMP::SolverSession::Execute`, `GMP::SolverSession::ExecutionStatus`, `GMP::SolverSession::Interrupt` and `GMP::SolverSession::WaitForSingleCompletion`.

---

## GMP::SolverSession::WaitForSingleCompletion

The routine `GMP::SolverSession::WaitForSingleCompletion` has a set of objects as its input. The set of objects may contain solver sessions that are asynchronous executing and events. This routine lets AIMMS wait until one of the solver sessions has completed its asynchronous execution or one of the events gets activated, and it returns the completed object.

```
GMP::SolverSession::WaitForSingleCompletion(  
    Objects          ! (input) a set of objects  
)
```

### Arguments:

*Objects*

A subset of `AllSolverSessionCompletionObjects`.

### Return value:

An element in the set `AllSolverSessionCompletionObjects`.

### Remarks:

- This routine ignores solver sessions that are not asynchronous executing but using the procedure `GMP::SolverSession::Execute`.
- This routine will return immediately if one of the objects is a solver session that has execution status 'Finished'.

### See also:

The routines `GMP::Event::Create`, `GMP::Event::Set`, `GMP::SolverSession::AsynchronousExecute`, `GMP::SolverSession::Execute`, `GMP::SolverSession::ExecutionStatus`, `GMP::SolverSession::Interrupt` and `GMP::SolverSession::WaitForCompletion`.