```
Index: chap-tools/prof.tex
===================================================================
--- chap-tools/prof.tex  (revision 1772)
+++ chap-tools/prof.tex  (revision 1911)
@@ -20,8 +20,8 @@
 supplied and work as expected, but it is not possible to replace at
 link time the \mpiskipfunc{MPI\_} version with a user-defined
version.

-For Fortran, the different support methods cause several linker
names.
-Therefore, several profiling routines (with these linker names)
+For Fortran, the different support methods cause several specific
procedure names.
+Therefore, several profiling routines (with these specific procedure
names)
 are needed for each Fortran \MPI/ routine, as described in
 \sectionref{sec:f90:linker-names}.

@@ -140,7 +140,7 @@
 \funcarg{\IN}{level}{Profiling level (integer)}
 \end{funcdef}
 \mpibind{MPI\_Pcontrol(const~int~level, \ldots)}
-\mpifnewbind{MPI\_Pcontrol(level) BIND(C) \fargs INTEGER,
INTENT(IN) :: level}
+\mpifnewbind{MPI\_Pcontrol(level) \fargs INTEGER, INTENT(IN) ::
level}
 \mpifbind{MPI\_PCONTROL(LEVEL)\fargs INTEGER LEVEL}
 \mpicppemptybind{MPI::Pcontrol(const int~level, \ldots)}{void}

@@ -368,7 +368,7 @@
 The different Fortran support methods and possible options
 for the support of subarrays (depending on whether the compiler
 can support \code{TYPE(*), DIMENSION(..)} choice buffers)
-imply different linker names for the same Fortran \MPI/ routine.
+imply different specific procedure names for the same Fortran \MPI/
routine.
 The rules and implications for the profiling interface are
 described in
 \sectionref{sec:f90:linker-names}.
Index: chap-tools/mpit.tex
===================================================================
--- chap-tools/mpit.tex  (revision 1772)
+++ chap-tools/mpit.tex  (revision 1911)
@@ -29,6 +29,24 @@
 descriptions about their meaning, and to access and, if appropriate,
to alter
 their values.

+Variables and categories across connected processes with equivalent
```

```
names are
+required to have the same meaning (see the definition of
``equivalent'' as related
+to strings in Section \ref{sec:mpit:strings}).
+Furthermore, enumerations with equivalent names across connected
processes are required to have the same meaning, but are allowed to
comprise different enumeration items.
+Enumeration items that have equivalent names across connected
processes in enumerations with the same meaning must also have the
same meaning.
+In order for variables and categories to have the same meaning,
routines in the tools information interface that return details for
those variables and categories have requirements on what parameters
+must be identical.
+These requirements are specified in
+their respective sections.
+
+\begin{rationale}
+The intent of requiring the same meaning for entities with equivalent
names is to enforce consistency
+across connected processes.
+For example, variables describing the number of packets sent
+on different types of network devices should have different names to
reflect their
+potentially different meanings.
+\end{rationale}
+
 The \MPI/ tool information interface can be used independently from
 the \MPI/ communication functionality. In particular, the routines of
this interface can be called
 before \mpifunc{MPI\_INIT} (or equivalent) and after \mpifunc{MPI
\_FINALIZE}.
@@ -48,10 +66,11 @@
 Since the \MPI/ tool information interface primarily focuses on tools
and
 support libraries,
 \MPI/ implementations are only required to provide C bindings
-for functions introduced in this section.  Except where otherwise
noted, all conventions
+for functions and constants
+introduced in this section.  Except where otherwise noted, all
conventions
 and principles governing the C bindings of the \MPI/ API also apply
to
 the \MPI/ tool information interface, which is available by including
-the \code{mpi.h} header file. All routines in this interface have
local semantics.
+the {\sf mpi.h} header file. All routines in this interface have
local semantics.
```

```
 \begin{users}
 The number and type of control variables and performance variables
can
@@ -58,10 +77,9 @@
 vary between \MPI/ implementations, platforms and different
 builds of the same implementation on the same platform as well as
 between runs. Hence, any
-application relying on a particular variable will not be portable.
Further,
-there is no guarantee that number of variables, variable indices,
-and variable names are the
-same across processes.
+application relying on a particular variable will not be portable.
+Further, there is no guarantee that the number of variables and
+variable indices are the same across connected processes.

 This interface is primarily intended for performance monitoring
tools,
 support tools, and libraries controlling the application's
@@ -172,11 +190,13 @@

 \subsection{Convention for Returning Strings}
 \label{sec:mpit:strings}
+
 Several \MPI/ tool information interface functions return one or more
-strings. These functions
-have two arguments for each string to be returned: an OUT parameter
+strings.
+These functions
+have two arguments for each string to be returned: an \OUT\ parameter
 that identifies a pointer to the buffer in which the string will be
-returned, and an IN/OUT parameter to pass the length of the buffer.
+returned, and an \IN/\OUT\ parameter to pass the length of the
buffer.
 The user is responsible for the memory allocation of the buffer and
 must pass the size of the buffer ($n$) as the length argument.  Let
 $n$ be the length value specified to the function.  On return, the
@@ -192,7 +212,15 @@
 argument, the buffer argument is ignored and nothing is returned.


+\MPI/ implementations behave as if they have an internal character
+array that is copied to the output character array supplied by the
user.
+Such output strings are defined to be equivalent if their notional
+source-internal character arrays are identical (up to and including
+the null terminator), even if the output string is truncated due
+to a small input length parameter $n$.

+
+
 \subsection{Initialization and Finalization}
 \label{sec:mpit:init}

@@ -229,13 +257,13 @@
 Section~\ref{sec:ei-threads}.

 The \MPI/ specification does not require all \MPI/ processes to exist
-before the call to \mpifunc{MPI\_INIT}.  If the \MPI/ tool
information
+before the call to \func{MPI\_INIT}.  If the \MPI/ tool information
 interface is used before
-\mpifunc{MPI\_INIT} has been
+\func{MPI\_INIT} has been
 called,
 the user is responsible for ensuring that the \MPI/ tool information
interface is initialized on all processes it is used in.
 Processes created by the \MPI/ implementation during
-\mpifunc{MPI\_INIT} inherit the status of the \MPI/ tool information
interface
+\func{MPI\_INIT} inherit the status of the \MPI/ tool information
interface
 (whether it is
 initialized or not as well as all active sessions and handles) from
the process
 from which they are created.
@@ -285,8 +313,8 @@
 to \mpifunc{MPI\_T\_INIT\_THREAD}.

 At the end of the program execution, unless \mpifunc{MPI\_ABORT} is
-called, an application must have called \mpifunc{MPI\_T\_INIT
\_THREAD}
-and \mpifunc{MPI\_T\_FINALIZE} an equal number of times.
+called, an application must have called \func{MPI\_T\_INIT\_THREAD}
+and \func{MPI\_T\_FINALIZE} an equal number of times.



@@ -300,12 +328,12 @@
 initialization of the \MPI/ tool information interface is separate
 from the initialization of
 \MPI/, \MPI/ tool information interface routines can be called before
-\mpifunc{MPI\_INIT}. Consequently, these routines can
-also use \MPI/ datatypes before \mpifunc{MPI\_INIT}. Therefore,
+\func{MPI\_INIT}. Consequently, these routines can
+also use \MPI/ datatypes before \func{MPI\_INIT}. Therefore,
 within the context of the \MPI/ tool information interface, it is
 permissible to use a subset of
 \MPI/ datatypes as specified below before a call to
-\mpifunc{MPI\_INIT} (or equivalent).
```

```
+\func{MPI\_INIT} (or equivalent).



@@ -340,17 +368,27 @@
 \end{rationale}

 The \MPI/ tool information interface only relies on a subset of the
basic \MPI/
-datatypes and does not use any derived \MPI/ datatypes.
+datatypes and does not use any derived \MPI/ datatypes.
 Table~\ref{table:tools:mpit:datatypes} lists all \MPI/ datatypes that
 can be returned by the \MPI/ tool information interface to represent
 its variables.

+The use of the datatype \const{MPI\_CHAR} in the \MPI/ tool
information
+interface implies a null-terminated character array, i.e., a string
in the C
+language.
+If a variable has type \const{MPI\_CHAR}, the value of the
\mpiarg{count}
+parameter returned by
+\mpifunc{MPI\_T\_CVAR\_HANDLE\_ALLOC} and \mpifunc{MPI\_T\_PVAR
\_HANDLE\_ALLOC}
+must  be large enough to include any valid value, including its
terminating null
+character.
+The contents of returned \const{MPI\_CHAR} arrays are only defined
from index 0 through
+the location of the first null character.

 \begin{rationale}
 The \MPI/ tool information interface requires a significantly simpler
 type system than \MPI/ itself. Therefore, only its required subset
 must be present before
-\mpifunc{MPI\_INIT} (or equivalent) and \MPI/ implementations
+\func{MPI\_INIT} (or equivalent) and \MPI/ implementations
 do not need to initialize the complete \MPI/
 datatype system.
 \end{rationale}
@@ -363,8 +401,8 @@
 We refer to
 this information in the following as an enumeration.  In this case,
the respective
 calls that provide additional metadata for each control or
performance
-variable, i.e., \mpifunc{MPI\_T\_CVAR\_GET\_INFO}
-(Section~\ref{sec:mpit:cvar}) and \mpifunc{MPI\_T\_PVAR\_GET\_INFO}
+variable, i.e., \func{MPI\_T\_CVAR\_GET\_INFO}
```

```
+(Section~\ref{sec:mpit:cvar}) and \func{MPI\_T\_PVAR\_GET\_INFO}
 (Section~\ref{sec:mpit:pvar}), return a handle of type
 \type{MPI\_T\_enum}\cdeclmainindex{MPI\_T\_enum}
 that can be passed to the following functions to
@@ -497,7 +535,9 @@
 information. An \MPI/ implementation is not allowed to alter any of
 the returned values.

+If any \gtype{\OUT} parameter to \mpifunc{MPI\_T\_CVAR\_GET\_INFO} is
a \consti{NULL} pointer, the implementation will ignore the parameter
and not return a value for the parameter.

+
 \textoutdesc{name}{the name of the control variable}

 If completed successfully, the routine is required to return a name
of
@@ -526,7 +566,7 @@
 \textoutdesc{desc}{a description of the control variable}

 Returning a description is optional. If an \mpi/ implementation
-does not to return a description, the first character for
+does not return a description, the first character for
 \mpiarg{desc} must be set to the null character and \mpiarg{desc
\_len}
 must be set to one at the return of this call.

@@ -554,6 +594,10 @@
 \mpiarg{scope} will be set to one of the constants listed in
 Table~\ref{table:tools:mpit:scopes}.

+If the name of a control variable is equivalent across connected
processes,
+the following \mpiarg{OUT}  parameters must be identical:
\mpiarg{verbosity},
+\mpiarg{datatype}, \mpiarg{enumtype}, \mpiarg{bind}, and
\mpiarg{scope}.
+The returned description must be equivalent.

 \begin{table}[h]
 \begin{center}
@@ -585,7 +629,34 @@
 \end{users}


+\begin{funcdef}{MPI\_T\_CVAR\_GET\_INDEX(name, cvar\_index)}
+\funcarg{\IN}{name}{name of the control variable (string)}
+\funcarg{\OUT}{cvar\_index}{index of the control variable (integer)}
+\end{funcdef}
```

```
+\mpibind{MPI\_T\_cvar\_get\_index(const char *name, int *cvar
\_index)}
+
+\mpifunc{MPI\_T\_CVAR\_GET\_INDEX} is a function for retrieving
+the index of a control variable given a known variable name. The
+\mpiarg{name} parameter is provided by the caller, and \mpiarg{cvar
\_index}
+is returned by the \MPI/ implementation. The \mpiarg{name} parameter
is a string
+terminated with a null character.
+
+This routine returns \const{MPI\_SUCCESS} on success and returns
+\const{MPI\_T\_ERR\_INVALID\_NAME} if \mpiarg{name} does not match
the name of
+any control variable provided by the implementation at the time of
the call.
+
+\begin{rationale}
+This routine is provided to enable fast retrieval of control
variables by a tool,
+assuming it knows the name of the variable for which it is looking.
+The number of variables exposed by the implementation can change over
time,
+so it is not possible for the tool to simply iterate over the list of
variables
+once at initialization. Although using \MPI/ implementation specific
variable
+names is not portable across \MPI/ implementations, tool developers
may choose to
+take this route for lower overhead at runtime because the tool will
not have to
+iterate over the entire set of variables to find a specific one.
+\end{rationale}
+
 \subsubsection{Example: Printing All Control Variables}

 \begin{example}
@@ -622,7 +693,7 @@
             &verbose, &datatype, NULL,
             NULL, NULL, /*no description */
             &bind, &scope);
-    if (err!=MPI_SUCCESS) return err;
+    if (err!=MPI_SUCCESS || err!=MPI_T_ERR_INVALID_INDEX) return err;
     printf("Var %i: %s\n", i, name);
   }

@@ -686,14 +757,10 @@

 The value of \mpiarg{cvar\_index} should
 be in the range $0$ to $num\_cvar-1$, where $num\_cvar$ is the number
```

```
of available control
-variables as determined from a prior call to \mpifunc{MPI\_T\_CVAR
\_GET\_NUM}.
+variables as determined from a prior call to \func{MPI\_T\_CVAR\_GET
\_NUM}.
 The type of the \MPI/ object it references must be consistent with
the type
-returned in the \mpiarg{bind} argument in a prior call to
\mpifunc{MPI\_T\_CVAR\_GET\_INFO}.
+returned in the \mpiarg{bind} argument in a prior call to \func{MPI
\_T\_CVAR\_GET\_INFO}.

-In the case that the \mpiarg{bind} argument returned by
-\mpifunc{MPI\_T\_CVAR\_GET\_INFO} equals \const{MPI\_T\_BIND\_NO
\_OBJECT}, the argument
-\mpiarg{obj\_handle} is ignored.
-
 \begin{funcdef}{MPI\_T\_CVAR\_HANDLE\_FREE(handle)}
 \funcarg{\INOUT}{handle}{handle to be freed (handle)}
 \end{funcdef}
@@ -702,7 +769,7 @@
 \mpibind{MPI\_T\_cvar\_handle\_free(MPI\_T\_cvar\_handle *handle)}

 When a handle is no longer needed, a user of the \MPI/ tool
information interface should call
-\mpifunc{MPI\_T\_CVAR\_HANDLE\_FREE} to free the handle and the
+\func{MPI\_T\_CVAR\_HANDLE\_FREE} to free the handle and the
 associated resources in the \MPI/ implementation.  On a successful
 return, \MPI/ sets the handle to \const{MPI\_T\_CVAR\_HANDLE\_NULL}.

@@ -717,9 +784,10 @@
 \cdeclindex{MPI\_T\_cvar\_handle}
 \mpibind{MPI\_T\_cvar\_read(MPI\_T\_cvar\_handle handle, void* buf)}

-This routine queries the value of the control
+This routine queries the value of a control
 variable identified by the argument \mpiarg{handle} and stores the
-result in the buffer identified by the parameter \mpiarg{buf}.  The
+result in the buffer identified by the parameter \mpiarg{buf}.
+The
 user must ensure that the buffer is of the appropriate
 size to hold the entire value of the control variable (based on the
 returned datatype and count from prior corresponding calls to
@@ -735,7 +803,8 @@

 This routine sets the value of the control variable
 identified by the argument \mpiarg{handle} to the data stored in the
-buffer identified by the parameter \mpiarg{buf}.  The user must
ensure that the
+buffer identified by the parameter \mpiarg{buf}.
```

```
+The user must ensure that the
 buffer is of the appropriate size to
 hold the entire value of the control variable (based on the returned
 datatype and count from prior corresponding calls to
@@ -761,7 +830,6 @@
 \const{MPI\_T\_ERR\_CVAR\_SET\_NEVER}, if the variable cannot be set
for the
 remainder of the application's execution.

-
 \subsubsection{Example: Reading the Value of a Control Variable}

 \begin{example}
@@ -807,7 +875,7 @@
 performance variables provided by the \MPI/
 implementation. Performance variables provide insight into \MPI/
 implementation specific internals and can represent information such
-as the state of the \MPI/ implementation (e.g., waiting blocked,
+as the state of the MPI implementation (e.g., waiting blocked,
 receiving, not active), aggregated timing data for submodules, or
 queue sizes and lengths.

@@ -827,7 +895,7 @@
 Each performance variable is associated with a class that describes
 its basic semantics, possible datatypes, basic behavior, its starting
value, whether it can
 overflow, and when and
-how an \MPI/ implementation can change the variable's value.  The
starting value is
+how an MPI implementation can change the variable's value.  The
starting value is
 the value that is assigned to the variable the first time that it is
used or
 whenever it is reset.

@@ -868,14 +936,16 @@
 value is set.  \MPI/ implementations must ensure that variables of
this class cannot overflow.

 \item \const{MPI\_T\_PVAR\_CLASS\_SIZE}\\
-A performance variable in this class represents a value that is the
fixed
-size of  a resource.  Values returned from variables in
+A performance variable in this class represents a value that is the
+size of a resource.  Values returned from variables in
 this class are non-negative and represented by one of the following
 datatypes: \const{MPI\_UNSIGNED}, \const{MPI\_UNSIGNED\_LONG},
\const{MPI\_UNSIGNED\_LONG\_LONG}, \const{MPI\_DOUBLE}.  The starting
-value is the current utilization level of the resource at the time
that the
```

```
+value is the current size of the
+resource at the time that the
 starting value is set.  \MPI/ implementations must ensure that
variables of this class cannot overflow.



+
 \item \const{MPI\_T\_PVAR\_CLASS\_PERCENTAGE}\\
 The value of a performance variable in this class represents the
 percentage utilization of a finite resource.  The value of a variable
@@ -893,7 +963,7 @@
 the variable.  It can be represented by one of the
 following datatypes: \const{MPI\_UNSIGNED}, \const{MPI\_UNSIGNED
\_LONG}, \const{MPI\_UNSIGNED\_LONG\_LONG}, \const{MPI\_DOUBLE}.  The
 starting value is the current utilization level of the resource at
the
-time that the starting value is set.  \MPI/ implementations must
ensure that variables of this class cannot
+time that the variable is started or reset.  \MPI/ implementations
must ensure that variables of this class cannot
 overflow.

 \item \const{MPI\_T\_PVAR\_CLASS\_LOWWATERMARK}\\
@@ -903,7 +973,7 @@
 of the variable.  It can be represented by one of
 the following datatypes: \const{MPI\_UNSIGNED}, \const{MPI\_UNSIGNED
\_LONG}, \const{MPI\_UNSIGNED\_LONG\_LONG}, \const{MPI\_DOUBLE}.  The
 starting value is the current utilization level of the resource at
the
-time that the starting value is set.  \MPI/ implementations must
ensure that variables of this class cannot
+time that the variable is started or reset.  \MPI/ implementations
must ensure that variables of this class cannot
 overflow.

 \item \const{MPI\_T\_PVAR\_CLASS\_COUNTER}\\
@@ -941,7 +1011,7 @@
 one of the following datatypes: \const{MPI\_UNSIGNED}, \const{MPI
\_UNSIGNED\_LONG}, \const{MPI\_UNSIGNED\_LONG\_LONG},
 \const{MPI\_DOUBLE}.  The starting value for variables of this class
is 0.
 If the type \const{MPI\_DOUBLE} is used, the units that represent
time
-in this datatype must match the units used by \mpifunc{MPI\_WTIME}.
+in this datatype must match the units used by \func{MPI\_WTIME}.
 Otherwise, the time units should be documented, e.g., in the
description
 returned by \mpifunc{MPI\_T\_PVAR\_GET\_INFO}.
 Variables of this class can overflow.
@@ -996,7 +1066,7 @@
```

```
 \funcarg{\OUT}{enumtype}{optional descriptor for enumeration
information (handle)}
 \textoutargs{desc}{a description of the performance variable}
 \funcarg{OUT}{bind}{type of \MPI/ object to which this variable must
be bound (integer)}
-\funcarg{\OUT}{readonly}{flag indicating whether the variable can be
\flushline % fix for margin
+\funcarg{\OUT}{readonly}{flag indicating whether the variable can be
\hfill\hbox{}\linebreak % fix for margin
 written/reset (integer)}
 \funcarg{\OUT}{continuous}{flag indicating whether the variable can
be started and stopped or is continuously active (integer)}
 \funcarg{\OUT}{atomic}{flag indicating whether the variable can be
atomically read and reset (integer)}
@@ -1012,6 +1082,7 @@
 information. An \MPI/ implementation is not allowed to alter any of
 the returned values.

+If any \gtype{\OUT} parameter to \mpifunc{MPI\_T\_PVAR\_GET\_INFO} is
a \consti{NULL} pointer, the implementation will ignore the parameter
and not return a value for the parameter.

 \textoutdesc{name}{the name of the performance variable}
 If completed successfully, the routine is required to return a name
of
@@ -1045,10 +1116,10 @@
 be used to gather more information as described in Section~
\ref{sec:types}.
 Otherwise, \mpiarg{enumtype} is set to \const{MPI\_T\_ENUM\_NULL}.
 If the datatype is not \const{MPI\_INT} or the argument
\mpiarg{enumtype} is the
-null pointer, no emumeration type is returned.
+null pointer, no enumeration type is returned.

 Returning a description is optional. If an \mpi/ implementation
-does not to return a description, the first character for
+does not return a description, the first character for
 \mpiarg{desc} must be set to the null character and \mpiarg{desc
\_len}
 must be set to one at the return from this function.

@@ -1071,7 +1142,45 @@
 which the call sets \mpiarg{atomic} to one can be used in
 a call to \mpifunc{MPI\_T\_PVAR\_READRESET}.

+If a performance variable has an equivalent name
+and has the same class across connected processes,
+the following \mpiarg{OUT}  parameters must be identical:
\mpiarg{verbosity},
+\mpiarg{varclass},
+\mpiarg{datatype}, \mpiarg{enumtype},
+\mpiarg{bind}, \mpiarg{readonly}, \mpiarg{continuous}, and
\mpiarg{atomic}.
+The returned description must be equivalent.

+\begin{funcdef}{MPI\_T\_PVAR\_GET\_INDEX(name, var\_class, pvar
\_index)}
+\funcarg{\IN}{name}{the name of the performance variable (string)}
+\funcarg{\IN}{var\_class}{the class of the performance variable
(integer)}
+\funcarg{\OUT}{pvar\_index}{the index of the performance variable
(integer)}
+\end{funcdef}
+
+\mpibind{MPI\_T\_pvar\_get\_index(const char *name, int var\_class,
int *pvar\_index)}
+
+\mpifunc{MPI\_T\_PVAR\_GET\_INDEX} is a function for retrieving
+the index of a performance variable given a known variable name and
class. The
+\mpiarg{name} and \mpiarg{var\_class} parameters are provided by the
caller, and \mpiarg{pvar\_index}
+is returned by the \MPI/ implementation. The \mpiarg{name} parameter
is a string
+terminated with a null character.
+
+This routine returns \const{MPI\_SUCCESS} on success and returns
+\const{MPI\_T\_ERR\_INVALID\_NAME} if \mpiarg{name} does not match
the
+name of any performance variable provided by the implementation
+at the time of the call.
+
+\begin{rationale}
+This routine is provided to enable fast retrieval of performance
+variables by a tool, assuming it knows the name of the variable for
+which it is looking. The number of variables exposed by the
implementation
+can change over time, so it is not possible for the tool to simply
iterate
+over the list of variables once at initialization. Although using
\MPI/
+implementation specific variable names is not portable across \MPI/
+implementations, tool developers may choose to take this route for
lower
+overhead at runtime because the tool will not have to iterate over
the
+entire set of variables to find a specific one.
+\end{rationale}
+
 \subsubsection{Performance Experiment Sessions}
 \label{sec:mpit:sessions}

@@ -1153,14 +1262,16 @@
 The value of index
 should be in the range $0$ to $\mpiarg{num\_pvar}-1$, where $
\mpishortarg{num\_pvar}$ is the number of
 available performance variables as determined from a prior call to
-\mpifunc{MPI\_T\_PVAR\_GET\_NUM}.  The type of the \MPI/ object it
references must be consistent with the type
-returned in the \mpiarg{bind} argument in a prior call to
\mpifunc{MPI\_T\_PVAR\_GET\_INFO}.
+\func{MPI\_T\_PVAR\_GET\_NUM}.  The type of the \MPI/ object it
references must be consistent with the type
+returned in the \mpiarg{bind} argument in a prior call to \func{MPI
\_T\_PVAR\_GET\_INFO}.

-In the case the \mpiarg{bind} argument equals \const{MPI\_T\_BIND\_NO
\_OBJECT}, the argument
-\mpiarg{obj\_handle} is ignored.
+For all routines in the rest of this section that take both
\mpiarg{handle}
+and \mpiarg{session} as \IN\ arguments, if the \mpiarg{handle}
argument passed in is not associated with the \mpiarg{session}
argument,
+\const{MPI\_T\_ERR\_INVALID\_HANDLE} is returned.



+
 \begin{funcdef}{MPI\_T\_PVAR\_HANDLE\_FREE(session, handle)}
 \funcarg{\IN}{session}{identifier of performance experiment session
(handle)}
 \funcarg{\INOUT}{handle}{handle to be freed (handle)}
@@ -1171,7 +1282,7 @@
 \mpibind{MPI\_T\_pvar\_handle\_free(MPI\_T\_pvar\_session session,
MPI\_T\_pvar\_handle *handle)}

 When
-a handle is no longer needed, a user of the \MPI/ tool information
interface should call \mpifunc{MPI\_T\_PVAR\_HANDLE\_FREE}
+a handle is no longer needed, a user of the \MPI/ tool information
interface should call \func{MPI\_T\_PVAR\_HANDLE\_FREE}
 to free the handle in the session identified
 by the parameter \mpiarg{session}
 and the associated resources in the \MPI/  implementation.
@@ -1195,7 +1306,7 @@

 \cdeclindex{MPI\_T\_pvar\_handle}
 \cdeclindex{MPI\_T\_pvar\_session}
-\mpibind{MPI\_T\_pvar\_start(MPI\_T\_pvar\_session session, MPI\_T
\_pvar\_handle handle)}
+ \mpibind{MPI\_T\_pvar\_start(MPI\_T\_pvar\_session session, MPI\_T
\_pvar\_handle handle)}

 This functions starts the performance variable with the handle
 identified by the parameter \mpiarg{handle} in the session identified
@@ -1205,8 +1316,10 @@
 \mpiarg{handle}, the \MPI/ implementation attempts to start all
variables
 within the session identified by the parameter \mpiarg{session} for
 which handles have been allocated.  In this case, the routine returns
-\const{MPI\_SUCCESS} if all variables are started successfully,
-otherwise \const{MPI\_T\_ERR\_PVAR\_NO\_STARTSTOP} is returned.
Continuous
+\const{MPI\_SUCCESS} if all variables are started successfully
+(even if there are no non-continuous variables to be started),
+otherwise \const{MPI\_T\_ERR\_PVAR\_NO\_STARTSTOP} is returned.
+Continuous
 variables and variables that are already started are ignored when
 \const{MPI\_T\_PVAR\_ALL\_HANDLES} is specified.

@@ -1229,8 +1342,10 @@
 variables within the session identified by the parameter
 \mpiarg{session} for which handles have been allocated.  In this
case,
 the routine returns \const{MPI\_SUCCESS} if all variables are stopped
-successfully, otherwise \const{MPI\_T\_ERR\_PVAR\_NO\_STARTSTOP} is
-returned. Continuous variables and variables that are already stopped
+successfully
+(even if there are no non-continuous variables to be stopped),
+otherwise \const{MPI\_T\_ERR\_PVAR\_NO\_STARTSTOP} is returned.
+Continuous variables and variables that are already stopped
 are ignored when \const{MPI\_T\_PVAR\_ALL\_HANDLES} is specified.


@@ -1246,10 +1361,11 @@
 \cdeclindex{MPI\_T\_pvar\_session}
 \mpibind{MPI\_T\_pvar\_read(MPI\_T\_pvar\_session session, MPI\_T
\_pvar\_handle handle, void* buf)}

-The \mpifunc{MPI\_T\_PVAR\_READ} call queries the value of the
+The \func{MPI\_T\_PVAR\_READ} call queries the value of the
 performance variable with the handle \mpiarg{handle} in the session
 identified by the parameter \mpiarg{session} and stores the result in
-the buffer identified by the parameter \mpiarg{buf}.  The user is
+the buffer identified by the parameter \mpiarg{buf}.
+The user is
 responsible to ensure that the buffer is of the appropriate size to
 hold the entire value of the performance variable (based on the
 datatype and count returned by the corresponding previous calls to
```

```
@@ -1257,7 +1373,7 @@

 The
 constant \const{MPI\_T\_PVAR\_ALL\_HANDLES} cannot be used as an
argument
-for the  function \mpifunc{MPI\_T\_PVAR\_READ}.
+for the  function \func{MPI\_T\_PVAR\_READ}.

 \begin{funcdef}{MPI\_T\_PVAR\_WRITE(session,handle, buf)}
 \funcarg{\IN}{session}{identifier of performance experiment session
(handle)}
@@ -1270,10 +1386,11 @@
 \mpibind{MPI\_T\_pvar\_write(MPI\_T\_pvar\_session session, MPI\_T
\_pvar\_handle handle, const void* buf)}


-The \mpifunc{MPI\_T\_PVAR\_WRITE} call attempts to write the value of
the
+The \func{MPI\_T\_PVAR\_WRITE} call attempts to write the value of
the
 performance variable with the handle identified by the parameter
 \mpiarg{handle} in the session identified by the parameter
-\mpiarg{session}. The value to be written is passed in the buffer
+\mpiarg{session}.
+The value to be written is passed in the buffer
 identified by the parameter \mpiarg{buf}.  The user must
 ensure that the buffer is of the appropriate size to hold the entire
 value of the performance variable (based on the
@@ -1285,9 +1402,8 @@
 \const{MPI\_T\_ERR\_PVAR\_NO\_WRITE}.

 The constant \const{MPI\_T\_PVAR\_ALL\_HANDLES} cannot be used as an
argument
-for the  function \mpifunc{MPI\_T\_PVAR\_WRITE}.
+for the  function \func{MPI\_T\_PVAR\_WRITE}.

-
 \begin{funcdef}{MPI\_T\_PVAR\_RESET(session, handle)}
 \funcarg{\IN}{session}{identifier of performance experiment session
(handle)}
 \funcarg{\IN}{handle}{handle of a performance variable (handle)}
@@ -1298,7 +1414,7 @@
 \mpibind{MPI\_T\_pvar\_reset(MPI\_T\_pvar\_session session, MPI\_T
\_pvar\_handle handle)}


-The \mpifunc{MPI\_T\_PVAR\_RESET} call sets the performance variable
with
+The \func{MPI\_T\_PVAR\_RESET} call sets the performance variable
with
```

```
 the handle identified by the parameter \mpiarg{handle} to its
starting
 value specified in Section~\ref{sec:tools:mpit:classes}. If it is not
 possible to change the variable, the function returns
@@ -1308,8 +1424,10 @@
 \MPI/ implementation attempts to reset all variables
 within the session identified by the parameter \mpiarg{session} for
 which handles have been allocated.  In this case, the routine returns
-\const{MPI\_SUCCESS} if all variables are reset successfully,
-otherwise \const{MPI\_T\_ERR\_PVAR\_NO\_WRITE} is returned. Read-only
variables
+\const{MPI\_SUCCESS} if all variables are reset successfully
+(even if there are no valid handles or all are read-only),
+otherwise \const{MPI\_T\_ERR\_PVAR\_NO\_WRITE} is returned.
+Read-only variables
 are ignored when \const{MPI\_T\_PVAR\_ALL\_HANDLES} is specified.


@@ -1323,28 +1441,27 @@
 \cdeclindex{MPI\_T\_pvar\_session}
 \mpibind{MPI\_T\_pvar\_readreset(MPI\_T\_pvar\_session session, MPI
\_T\_pvar\_handle handle, void* buf)}

-This call atomically combines the functionality of \mpifunc{MPI\_T
\_PVAR\_READ} and
-\mpifunc{MPI\_T\_PVAR\_RESET} with the same semantics as if these two
-calls were called separately. If atomic operations on this variable
are not
+This call atomically combines the functionality of \func{MPI\_T\_PVAR
\_READ} and
+\func{MPI\_T\_PVAR\_RESET} with the same semantics as if these two
+calls were called separately.
+If atomic operations on this variable are not
 supported, this routine returns \const{MPI\_T\_ERR\_PVAR\_NO
\_ATOMIC}.

 The constant \const{MPI\_T\_PVAR\_ALL\_HANDLES} cannot be used as an
-argument for the  function \mpifunc{MPI\_T\_PVAR\_READRESET}.
+argument for the  function \func{MPI\_T\_PVAR\_READRESET}.

-
-
 \begin{implementors}
-Sampling-based tools rely on the ability to call the \MPI/
+Sampling-based tools rely on the ability to call the MPI
 tool information interface, in particular routines to start, stop,
 read, write and reset performance variables, from any program
 context, including asynchronous contexts such as signal handlers.
-\MPI/ implementations should strive, if possible in their particular
+MPI implementations should strive, if possible in their particular
```

```
 environment, to enable these usage scenarios for all or a subset of
the
 routines mentioned above. If implementing only a subset, the
 read, write, and reset routines are typically the most critical
-for sampling based tools. An \MPI/ implementation should clearly
+for sampling based tools. An MPI implementation should clearly
 document any restrictions on the program contexts in which
-the \MPI/ tool information interface can be used. Restrictions
+the MPI tool information interface can be used. Restrictions
 might include guaranteeing usage outside of all signals or
 outside a specific set of signals. Any restrictions could be
 documented, for example, through the description returned by
@@ -1370,7 +1487,7 @@
 The following example shows a sample tool to identify receive
 operations that occur during times with long message queues.  This
examples
 assumes that the \MPI/ implementation exports a variable with the
name
-\code{MPI\_T\_UMQ\_LENGTH} to represent the current length of
+``\texttt{MPI\_T\_UMQ\_LENGTH}'' to represent the current length of
 the unexpected message queue. The tool is implemented as a
 PMPI tool using the \MPI/ profiling interface.

@@ -1383,7 +1500,7 @@
 would have to be extended to have similar wrappers for all
 receive operations.

-\paragraph{Part 1 --- Initialization:}
+\paragraph{Part 1--- Initialization:}

 During initialization, the tool searches for the variable and, once
 the right index is found, allocates a session and a handle for the
@@ -1555,7 +1672,7 @@
 categories, but they are not allowed to remove variables from
 categories or change the order in which they are returned.

-The following function can be used to query the number of control
variables,
+The following function can be used to query the number of categories,
 $N$.

 \begin{funcdef}{MPI\_T\_CATEGORY\_GET\_NUM(num\_cat)}
@@ -1585,6 +1702,9 @@
 name must be unique with respect to all other names for
 categories used by the \MPI/ implementation.

+If any \gtype{\OUT} parameter to \mpifunc{MPI\_T\_CATEGORY\_GET
\_INFO} is a \consti{NULL} pointer, the implementation will ignore the
parameter and not return a value for the parameter.
+
+
 \textoutdesc{desc}{the description of the category}

 Returning a description is optional. If an \mpi/ implementation
@@ -1598,7 +1718,39 @@
 \mpiarg{num\_pvars}, and \mpiarg{num\_categories},
 respectively.

+If the name of a category is equivalent across connected
+processes, then the returned description must be equivalent.

+\begin{funcdef}{MPI\_T\_CATEGORY\_GET\_INDEX(name, cat\_index)}
+\funcarg{\IN}{name}{the name of the category (string)}
+\funcarg{\OUT}{cat\_index}{the index of the category (integer)}
+\end{funcdef}
+
+\mpibind{MPI\_T\_category\_get\_index(const char *name, int *cat
\_index)}
+
+\mpifunc{MPI\_T\_CATEGORY\_GET\_INDEX} is a function for
+retrieving the index of a category given a known category name. The
+\mpiarg{name} parameter is provided by the caller, and \mpiarg{cat
\_index}
+is returned by the \MPI/ implementation. The \mpiarg{name} parameter
+is a string terminated with a null character.
+
+This routine returns \const{MPI\_SUCCESS} on success and returns
+\const{MPI\_T\_ERR\_INVALID\_NAME} if \mpiarg{name} does not match
the
+name of any category provided by the implementation at the time of
the call.
+
+\begin{rationale}
+This routine is provided to enable fast retrieval of a category index
by
+a tool, assuming it knows the name of the category for which it is
looking.
+The number of categories exposed by the implementation can change
over time,
+so it is not possible for the tool to simply iterate over the list of
+categories once at initialization. Although using \MPI/
implementation
+specific category names is not portable across \MPI/ implementations,
+tool developers may choose to take this route for lower overhead at
runtime
+because the tool will not have to iterate over the entire set of
categories to
+find a specific one.
+\end{rationale}
+
+
 \begin{funcdef}{MPI\_T\_CATEGORY\_GET\_CVARS(cat\_index, len,
indices)}
 \funcarg{\IN}{cat\_index}{index of the category to be queried, in the
range $[0,N-1]$ (integer)}
 \funcarg{\IN}{len}{the length of the indices array (integer)}
@@ -1702,11 +1854,11 @@
 and cannot overlap with any other error codes or error
 classes returned by the \MPI/ implementation. Further, they shall be
 treated as \MPI/ error classes as defined in
-\sectionref{sec:ei-error-classes} and follow
+Section~\ref{sec:ei-error-classes} on page~\pageref{sec:ei-error-
classes} and follow
 the same rules and restrictions. In particular, they must satisfy:

 $$
-0 = \const{MPI\_SUCCESS} < \const{MPI\_T\_ERR\_\ldots} \leq
\const{MPI\_ERR\_LASTCODE}.
+0 = \const{MPI\_SUCCESS} < \const{MPI\_T\_ERR\_...} \leq \const{MPI
\_ERR\_LASTCODE}.
 $$

 \begin{rationale}
@@ -1725,6 +1877,7 @@
 \multicolumn{2}{|l|}{Return Codes for All Functions in the \MPI/ Tool
Information Interface}\\
 \hline
 \const{MPI\_SUCCESS} & Call completed successfully\\
+\const{MPI\_T\_ERR\_INVALID} & Invalid use of the interface or bad
parameter value(s) \\
 \const{MPI\_T\_ERR\_MEMORY} & Out of memory\\
 \const{MPI\_T\_ERR\_NOT\_INITIALIZED} & Interface not initialized\\
 \const{MPI\_T\_ERR\_CANNOT\_INIT} & Interface not in the state to be
initialized\\
@@ -1735,22 +1888,25 @@
 \mpifuncindex{MPI\_T\_ENUM\_GET\_ITEM}%
 }\\
 \hline
-\const{MPI\_T\_ERR\_INVALID\_INDEX} & The enumeration index is
invalid or has \\
-                                    & been deleted.\\
-\const{MPI\_T\_ERR\_INVALID\_ITEM}  & The item index queried is out
of range\\
-                                   & (for \mpifunc{MPI\_T\_ENUM\_GET
\_ITEM} only)\\
+\const{MPI\_T\_ERR\_INVALID\_INDEX} & The enumeration index is
invalid  \\
+\const{MPI\_T\_ERR\_INVALID\_ITEM}  & The item index queried is out
of range \\
+& (for \mpifunc{MPI\_T\_ENUM\_GET\_ITEM} only) \\
 \hline
 \hline
-\multicolumn{2}{|l|}{Return Codes for variable and category query
functions: {\mpiskipfunc{MPI\_T\_*\_GET\_INFO}}
+\multicolumn{2}{|l|}{Return Codes for Variable and Category Query
Functions: {\mpiskipfunc{MPI\_T\_*\_GET\_*}}
 \mpifuncindex{MPI\_T\_PVAR\_GET\_INFO}%
 \mpifuncindex{MPI\_T\_CVAR\_GET\_INFO}%
 \mpifuncindex{MPI\_T\_CATEGORY\_GET\_INFO}%
+\mpifuncindex{MPI\_T\_CVAR\_GET\_INDEX}%
+\mpifuncindex{MPI\_T\_PVAR\_GET\_INDEX}%
+\mpifuncindex{MPI\_T\_CATEGORY\_GET\_INDEX}%
 }\\
 \hline
 \const{MPI\_T\_ERR\_INVALID\_INDEX} & The variable or category index
is invalid\\
+\const{MPI\_T\_ERR\_INVALID\_NAME} & The variable or category name is
invalid\\
 \hline
 \hline
-%% BRONIS: Changed to use mpiskipfunc to be consistent with other
chapters
+
 \multicolumn{2}{|l|}{Return Codes for Handle Functions:
{\mpiskipfunc{MPI\_T\_*\_\textrm{\{}ALLOC$|$FREE\textrm{\}}}}
 \mpifuncindex{MPI\_T\_PVAR\_HANDLE\_ALLOC}%
 \mpifuncindex{MPI\_T\_CVAR\_HANDLE\_ALLOC}%
@@ -1758,7 +1914,7 @@
 \mpifuncindex{MPI\_T\_CVAR\_HANDLE\_FREE}%
 }\\
 \hline
-\const{MPI\_T\_ERR\_INVALID\_INDEX} & The variable index is invalid
or has been deleted\\
+\const{MPI\_T\_ERR\_INVALID\_INDEX} & The variable index is invalid \
\
 \const{MPI\_T\_ERR\_INVALID\_HANDLE} & The handle is invalid\\
 \const{MPI\_T\_ERR\_OUT\_OF\_HANDLES} & No more handles available\\
 \hline
@@ -1781,7 +1937,7 @@
 \hline
 \hline
 \multicolumn{2}{|l|}{Return Codes for Performance Variable Access and
Control:}\\
-%% BRONIS: Changed to use mpiskipfunc to be consistent with other
chapters
+
 \multicolumn{2}{|l|}{\mpiskipfunc{MPI\_T\_PVAR\_\textrm{\{}START$|
$STOP$|$READ$|$WRITE$|$RESET$|$READREST\textrm{\}}}
 \mpifuncindex{MPI\_T\_PVAR\_START}%
 \mpifuncindex{MPI\_T\_PVAR\_STOP}%
```

```
@@ -1793,11 +1949,9 @@
 \hline
 \const{MPI\_T\_ERR\_INVALID\_HANDLE} & The handle is invalid\\
 \const{MPI\_T\_ERR\_INVALID\_SESSION} & Session argument is not a
valid session\\
-%% BRONIS Changed ``can not'' to ``cannot''
 \const{MPI\_T\_ERR\_PVAR\_NO\_STARTSTOP} & Variable cannot be started
or stopped\\
                                  & (for \mpifunc{MPI\_T\_PVAR\_START}
and\\
                                  & \mpifunc{MPI\_T\_PVAR\_STOP})\\
-%% BRONIS Changed ``can not'' to ``cannot''
 \const{MPI\_T\_ERR\_PVAR\_NO\_WRITE} & Variable cannot be written or
reset\\
                              & (for \mpifunc{MPI\_T\_PVAR\_WRITE} and
\\
                              & \mpifunc{MPI\_T\_PVAR\_RESET}) \\
```