PADS Manual
(Version 2.01)

The PADS Project
www.padsproj.org

Last revised: May 10, 2009

Contents

Preface

Acknowledgments

We would like to thank Bala Krishnamurthy, Andrew Hume, David Poole, and Oliver Spatscheck for informative discussions about particular forms of ad hoc data and potentially useful tools for manipulating that data. We would like to thank Glenn Fowler and Phong Vo for their help in using the AST and SFIO libraries. We would like to thank Mary Fernandez, Ricardo Medel, and Yitzhak Mandelbaum for their assistance in integrating PADS and Galax. Xuan Zheng implemented the histogram library.

Chapter 1

Introduction

Vast amounts of useful data are stored and processed in ad hoc formats. Traditional databases and XML systems provide rich infrastructure for processing well-behaved data, but are of little help when dealing with ad hoc formats. Examples that we face at AT&T include call detail data [CFP+04], web server logs [KR01], netflows capturing internet traffic [net], log files characterizing IP backbone resource utilization, wire formats for legacy telecommunication billing systems, *etc.* Such data may simply require processing before it can be loaded into a data management system, or it may be too large or too transient to make such loading cost effective.

Processing ad hoc data can be challenging for a variety of reasons. First, ad hoc data typically arrives "as is": the analysts who receive it can only say "thank you," not request a more convenient format. Second, documentation for the format may not exist at all, or it may be out of date. A common phenomenon is for a field in a data source to fall into disuse. After a while, a new piece of information becomes interesting, but compatibility issues prevent data suppliers from modifying the shape of their data, so instead they hijack the unused field, often failing to update the documentation in the process.

Third, such data frequently contain errors, for a variety of reasons: malfunctioning equipment, race conditions on log entry [KR01], non-standard values to indicate "no data available," human error in entering data, unexpected data values, *etc.* The appropriate response to such errors depends on the application. Some applications require the data to be error free: if an error is detected, processing needs to stop immediately and a human must be alerted. Other applications can repair the data, while still others can simply discard erroneous or unexpected values. For some applications, errors in the data can be the most interesting part because they can signal where two systems are failing to communicate.

A fourth challenge is that ad hoc data sources can be high volume: AT&T's call-detail stream contains roughly 300 million calls per day requiring approximately 7GBs of storage space. Although this data is eventually archived in a database, analysts mine it profitably before such archiving [CP98, CP99]. More challenging, the Altair project at AT&T accumulates billing data at a rate of 250-300GB/day, with occasional spurts of 750GBs/day. Netflow data arrives from Cisco routers at rates over a gigabyte per second [CGJ+02]! Such volumes mean it must be possible to process the data without loading it all into memory at once.

Finally, before anything can be done with an ad hoc data source, someone has to produce a suitable parser for it. Today, people tend to use C or PERL for this task. Unfortunately, writing parsers this way is tedious and error-prone, complicated by the lack of documentation, convoluted encodings designed to save space, the need to produce efficient code, and the need to handle errors robustly to avoid corrupting down-stream data. Moreover, the parser writers' hard-won understanding of the data ends up embedded in parsing code, making long-term maintenance difficult for the original writers and sharing the knowledge with others nearly impossible.

The PADS system makes life easier for data analysts by addressing each of these concerns.[1] It provides a declarative data description language that permits analysts to describe the physical layout of their data, *as it is*. The language also permits analysts to describe expected semantic properties of their data so that deviations can be flagged as errors. The intent is to allow analysts to capture in a PADS description all that they know about a given data source and to provide the analysts with a library of useful routines in exchange.

PADS descriptions are concise enough to serve as documentation and flexible enough to describe most of the data formats we have seen in practice, including ASCII, binary, Cobol, and mixed data formats. The fact that useful software artifacts are generated from the descriptions provides strong incentive for keeping the descriptions current, allowing them to serve as living documentation.

Given a PADS description, the PADS compiler produces customizable C libraries and tools for parsing, manipulating, and summarizing the data. The core C library includes functions for reading the data, writing it back out in its original form, and accumulating statistical properties. Future releases will include support for pretty printing the data, writing it into a canonical XML form, and implementing the Galax data model, which allows user to query PADS data using XQuery.

The declarative nature of PADS descriptions facilitates the insertion of error handling code. The generated parsing code checks all possible error cases: system errors related to the input file, buffer, or socket; syntax errors related to deviations in the physical format; and semantic errors in which the data violates user constraints. Because these checks appear only in generated code, they do not clutter the high-level declarative description of the data source. The result of a parse is a pair consisting of a canonical in-memory representation of the data and a parse descriptor. The parse descriptor precisely characterizes both the syntactic and the semantic errors that occurred during parsing. This structure allows analysts to choose how to respond to errors in application-specific ways.

With such huge datasets, performance is critical. The PADS system addresses performance in a number of ways. First, we compile the PADS description rather than simply interpret it to reduce run-time overhead. Second, the generated parser provides multiple entry points, so the data consumer can choose the appropriate level of granularity for reading the data into memory to accommodate very large data sources. Finally, we parameterize library functions by *masks*, which allow data analysts to choose which semantic conditions to check at run-time, permitting them to specify all known properties in the source description without forcing all users of that description to pay the run-time cost of checking them.

Given the importance of the problem, it is perhaps surprising that more tools do not exist to solve it. XML and relational databases only help with data already in well-behaved formats. Lex and Yacc are both over- and under- kill. Overkill because the division into a lexer and a context free grammar is not necessary for many ad hoc data sources, and under-kill in that such systems require the user to build in-memory representations manually, support only ASCII sources, and don't provide extra tools. ASN.1 [Dub01] and related systems [asd] allow the user to specify an in-memory representation and generate an on-disk format, but this doesn't help when given a particular on-disk format. Existing ad hoc description languages [Bac02, MC98, erla] are steps in the right direction, but they focus on binary, error-free data and they do not provide auxiliary tools.

1.1 PADS language

A PADS description specifies the physical layout and semantic properties of an ad hoc data source. The language provides a type-based model: basic types describe atomic data such as integers, strings, dates, *etc.*, while structured types describe compound data built from simpler pieces.

The PADS library provides a collection of broadly useful base types. Examples include 8-bit unsigned integers (`Puint8`), 32-bit integers (`Pint32`), dates (`Pdate`), strings (`Pstring`), and IP addresses (`Pip`). Semantic conditions for such base types include checking that the resulting number fits in the indicated

[1] PADS is short for Processing Ad hoc Data Sources.

space, *i.e.*, 16-bits for `Pint16`. By themselves, these base types do not provide sufficient information to allow parsing because they do not specify how the data is coded, *i.e.*, in ASCII, EBCDIC, or binary. To resolve this ambiguity, PADS uses the *ambient* coding, which the programmer can set. By default, PADS uses ASCII. To specify a particular coding, the description writer can select base types which indicate the coding to use. Examples of such types include ASCII 32-bit integers (`Pa_int32`), binary bytes (`Pb_int8`), and EBCDIC characters (`Pe_char`). In addition to these types, users can define their own base types to specify more specialized forms of atomic data.

To describe more complex data, PADS provides a collection of structured types loosely based on C's type structure. In particular, PADS has **`Pstruct`**s, **`Punion`**s, and **`Parray`**s to describe record-like structures, alternatives, and sequences, respectively. **`Penum`**s describe a fixed collection of literals, while **`Popt`**s provide convenient syntax for optional data. Each of these types can have an associated predicate that indicates whether a value calculated from the physical specification is indeed a legal value for the type. For example, a predicate might require that two fields of a **`Pstruct`** are related or that the elements of a sequence are in increasing order. Programmers can specify such predicates using PADS expressions and functions, written using a C-like syntax. Finally, PADS **`Ptypedef`**s can be used to define new types that add further constraints to existing types.

PADS types can be parameterized by values. This mechanism serves both to reduce the number of base types and to permit the format and properties of later portions of the data to depend upon earlier portions. For example, the base type `Puint16_FW(:3:)` specifies an unsigned two byte integer physically represented by exactly three characters, while the type `Pstring(:' ':)` describes a string terminated by a space. Parameters can be used with compound types to specify the size of an array or which branch of a union should be taken.

As an example, consider the common log format for Web server logs. A typical record looks like the following:

```
207.136.97.49 - - [15/Oct/1997:18:46:51 -0700] "GET /tk/p.txt HTTP/1.0" 200 30
```

recording the IP address of the requester; either a dash or the owner of the TCP session; either a dash or the login of the requester; the date; the actual request, which consists of the HTTP method, the requested URL, the HTTP version number; a response code; and the number of bytes returned. A PADS type describing the request portion is

```
Pstruct http_request_t {
  '\"'; http_method_t   meth;     /- Method used during request
  ' ';  Pstring(:' ':)  req_uri;  /- Requested uri.
  ' ';  http_v_t        version : checkVersion(version, meth);
                                  /- HTTP version
  '\"';
};
```

This **`Pstruct`** uses (omitted) auxiliary types `http_method_t` and `http_v_t` to describe the HTTP method and version formats, respectively. It uses character literals (`"\""` and `' '`) to consume the quotes and spaces from the physical representation. The `version` field has a constraint predicate `checkVersion` which ensures that obsolete HTTP methods `LINK` and `UNLINK` are only used with HTTP version `1.0`.

1.2 Generated library

From a description, the PADS compiler generates a C library for parsing and manipulating the associated data source. Figure 1.1 shows the architecture of the PADS system. The output of the PADS compiler is linked with the PADS standard library and user code to form an executable. In addition to the generated library, the

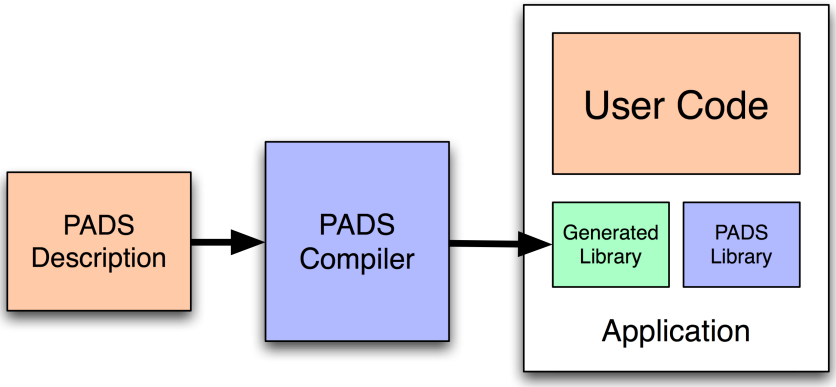

Figure 1.1: Architecture of PADS system.

PADS system provides template programs for accumulating, pretty-printing, and converting into XML data formats that have a particular format, namely, an optional header followed by a sequence of records.

From each type in a PADS description, the compiler generates

- an in-memory representation,
- a mask, which allows users to customize generated functions,
- a parse descriptor, which describes syntactic and semantic errors detected during parsing,
- parsing and printing functions, and
- a broad collection of utility functions.

The C declarations for the in-memory representation, the mask, and the parse descriptor all share the structure of the PADS type declaration. The mapping to C for each is straightforward: **`Pstruct`**s map to C structs with appropriately mapped fields, **`Punion`**s map to tagged unions coded as C structs with a tag field and an embedded union, **`Parray`**s map to a C struct with a length field and a pointer to a dynamically allocated sequence, **`Penums`** map to C enumerations, **`Poptions`** to tagged unions, and **`Ptypedef`**s to C typedefs. Masks include auxiliary fields to control behavior at the level of a structured type, and parse descriptors include extra fields to record the state of the parse, the number of detected errors, the error code of the first detected error, and the location of that error.

The parser takes a mask as an argument and returns an in-memory representation and a parse descriptor. The mask allows the user to specify which constraints the parser should check and which portions of the in-memory representation it should fill in. This control allows the description-writer to specify all known constraints about the data without worrying about the run-time cost of verifying potentially expensive constraints for time-critical applications.

Appropriate error-handling can be as important as processing error-free data. The parse descriptor marks which portions of the data contain errors and characterizes the detected errors. Depending upon the nature of the errors and the desired application, programmers can take the appropriate action: halting the program, discarding parts of the data, or repairing the errors. If the mask requests that a data item be verified and set, and if the parse descriptor indicates no error, then the in-memory representation satisfies the semantic constraints on the data.

Because we generate a parsing function for each type in a PADS description, we support multiple-entry point parsing, which allows us to accommodate larger-scale data. For a small file, programmers can define a PADS type that describes the entire file and use that type's parsing function to read the whole file with

one call. For larger-scale data, programmers can sequence calls to parsing functions that read manageable portions of the file, *e.g.*, reading a record at a time in a loop. The parsing code generated for **`Parrays`** allows users to choose between reading the entire array at once or reading it one element at a time, again to support parsing and processing very large data sources.

The ratio of the size of the data description to the size of the generated code gives a rough measure of the leverage of the declarative description. For the 68 line Sirius data description, the compiler yields a 1432 line `.h` file and a 6471 line `.c` file. This expansion comes from the extensive error checking in the generated parser and the number of generated utility functions.

1.3 Related work

There are many tools for describing data formats. For example, ASN.1 [Dub01] and ASDL [asd] are both systems for declaratively describing data and then generating libraries for manipulating that data. In contrast to PADS, however, both these systems specify the *logical* representation and automatically generate a *physical* representation. Although useful for many purposes, this technology does not help process data that arrives in predetermined, *ad hoc* formats.

More closely related work allows declarative descriptions of physical data [MC98, erlb, Bac02], motivated by parsing TCP/IP packets and JAVA jar-files. In contrast to our work, these systems only handle binary data and assume the data is error-free or halt parsing if an error is detected.

1.4 Terminology

We use the term *external* to refer to the physical data characterized by a PADS description. Such data might come from disk or over-the-wire for network applications.

1.5 Notation

In describing the syntax of various PADS expressions, we will use a BNF grammar. The syntax for pervasive features of the languages appears in Chapter 3. The syntax of a particular feature appears in the chapter describing that feature. We adopt the following conventions:

- **`Keywords`**
- [Expressions] are optional.
- *Non-terminals*

1.6 How do I read this manual?

In Chapter 2, we describe a sample use of the PADS system. It provides an overview of the data description language and illustrates the use of the generated library. Chapter 3 describes features of the PADS language common to all PADS types. Successive chapters assume familiarity with the material in this chapter. Chapter 4 describes built-in PADS base types, while Chapters 5 through 10 describe the structured types that PADS provides. These chapters describe the syntax and semantics of the PADS language and the core of the library generated for each such type declaration. In Chapter 14, we document how to use the PADS library, and in

Chapter 15 we explain the various ways in which the PADS library can be customized. Chapter 16 describes the generated accumulator library.

1.7 Getting the PADS system

Source code for PADS is available for download with a non-commercial use license from:

```
www.padsproj.org
```

The PADS distribution contains:

- This manual.
- Source code for the PADS system.
- README file describing installation procedure.
- Example directory with sample PADS description and client programs.

1.8 Using the PADS compiler

This section describes how to use the PADS compiler `padsc`. The simplest use of `padsc` is to compile a PADS source file to a C header and implementation file. The command

```
padsc my.p
```

will produce `my.h` and `my.c` files containing the generated library. By default, the compiler will also generate a file `my.xsd` containing the XSchema for the PADS canonical embedding of the data source into XML.

If the directory containing `my.p` has a subdirectory named `gen`, then the compiler will put the generated files in that subdirectory. Otherwise, it will put them in the same directory as the `my.p` file.

1.8.1 Command-line switches

Figure 1.2 lists the command-line options supported by the PADS compiler.

1.9 Practical notes

PADS uses the `Sfio` library for managing I/O streams. `Sfio` provides functionality similar to that of `Stdio`, the ANSI C Standard I/O library, but via a distinct interface. Sfio provides both source and binary `Stdio` emulation packages. `Sfio` is included in the PADS distribution. More information about `Sfio` is available from `www.research.att.com/\~gsf/download/ref/sfio/sfio.html`.

Please report any bugs in the PADS implementation or problems with this manual by electronic mail to `pads-dev@research.att.com`.

```
padsc  [-wnone] [-xsnone] [-anone] [-x] [-hist] [-cluster] [-r <string>] [-b
<string>] [-I <string> ...] [-D <string> ...] [-U <string> ...] [-t] [--help]
file...
-wnone  suppress write function generation
-xsnone suppress XSchema generation
-anone  suppress accumulator generation
-x      output Galax Data API
-hist   output histogram functions
-cluster        output cluster functions
-r      output directory
-b      add base type table
-I      augment include path
-D      add definition
-U      remove definition
-t      trace system commands

File extensions:
.p      PADS files
```

Figure 1.2: Command-line switches understood by the PADS compiler.

Chapter 2

Tutorial

In this chapter, we use examples to give an overview of PADS. The examples used in this chapter appear in the PADS distribution in the `pads/demo` directory.

2.1 Example data formats

We start by briefly describing the data formats we will use as running examples.

2.1.1 CLF: Common log format

One of the formats that web servers use to log client requests is the Common Log Format (CLF) [KR01]. Researchers use such logs to measure properties of web workloads and to evaluate protocol changes by "replaying" the user activity recorded in the log. This ASCII format consists of a sequence of records, each of which has seven fields: the host name or IP address of the client making the request, the account associated with the request on the client side, the name the user provided for authentication, the time of the request, the actual request, the HTTP response code, and the number of bytes returned as a result of the request. The actual request has three parts: the request method (*e.g.*, `GET`, `PUT`), the requested URI, and the protocol version. In addition, the second and third fields are often recorded only as a '-' character to indicate the server did not record the actual data. Figure 2.1 shows a couple of typical records.

2.1.2 Provisioning data

In the telecommunications industry, the term *provisioning* refers to the steps necessary to convert an order for phone service into the actual service. To track AT&T's provisioning process, the Sirius project compiles weekly summaries of the state of certain types of phone service orders. These ASCII summaries store the summary date and one record per order. Each order record contains a header followed by a sequence of events. The header has 13 pipe separated fields: the order number, AT&T's internal order number, the order version, four different telephone numbers associated with the order, the zip code of the order, a billing identifier, the order type, a measure of the complexity of the order, an unused field, and the source of the order data. Many of these fields are optional, in which case nothing appears between the pipe characters. The billing identifier may not be available at the time of processing, in which case the system generates a unique identifier, and prefixes this value with the string "no_ii" to indicate the number was generated. The event sequence represents the various states a service order goes through; it is represented as a new-line terminated, pipe separated list

```
207.136.97.49 - - [15/Oct/1997:18:46:51 -0700] "GET /tk/p.txt HTTP/1.0" 200 30
tj62.aol.com - - [16/Oct/1997:14:32:22 -0700] "POST /scpt/dd@grp.org/confirm HTTP/1.0" 200 941
```

Figure 2.1: Tiny example of web server log data.

```
0|1005022800
9152|9152|1|9735551212|0||9085551212|07988|no_ii152272|EDTF_6|0|APRL1|DUO|10|1000295291
9153|9153|1|0|0|0|0||152268|LOC_6|0|FRDW1|DUO|LOC_CRTE|1001476800|LOC_OS_10|1001649601
```

Figure 2.2: Tiny example of Sirius provisioning data.

of state, timestamp pairs. There are over 400 distinct states that an order may go through during provisioning. The sequence is sorted in order of increasing timestamps. Figure 2.2 shows a small example of this format.

2.2 PADS descriptions

Figure 2.3 gives the PADS description for CLF web server logs, while Figure 2.4 gives the description for the Sirius provisioning data. We use these examples to illustrate various features of the PADS language. In PADS descriptions, types are declared before they are used, so the type that describes the totality of the data source appears at the bottom of the description.

Pstructs describe fixed sequences of data with unrelated types. In the CLF description, the type declaration for `version_t` illustrates a simple **Pstruct**. It starts with a string literal that matches the constant `HTTP/` in the data source. It then has two unsigned integers recording the major and minor version numbers separated by the literal character `'.'`. PADS supports character, string, and regular expression literals, which are interpreted with the ambient character encoding. The type `request_t` similarly describes the request portion of a CLF record. In addition to physical format information, this **Pstruct** includes a semantic constraint on the `version` field. Specifically, it requires that obsolete methods `LINK` and `UNLINK` occur only under HTTP/1.1. This constraint illustrates the use of predicate functions and the fact that earlier fields are in scope during the processing of later fields, as the constraint refers to both the `meth` and `version` fields in the **Pstruct**. Chapter 5 describes **Pstruct**s in detail.

Punions describe variation in the data source. For example, the `client_t` type in the CLF description indicates that the first field in a CLF record can be either an IP address or a hostname. During parsing, the branches of a **Punion** are tried in order; the first branch that parses without error is taken. The `auth_id_t` type illustrates the use of a constraint: the branch `unauthorized` is chosen only if the parsed character is a dash. PADS also supports a *switched* union that uses a selection expression to determine the branch to parse. Typically, this expression depends upon already-parsed portions of the data source.

PADS provides **Parray**s to describe varying-length sequences of data all with the same type. The `eventSeq_t` declaration in the Sirius data description uses a **Parray** to characterize the sequence of events an order goes through during processing. This declaration indicates that the elements in the sequence have type `event_t`. It also specifies that the elements will be separated by vertical bars, and that the sequence will be terminated by an end-of-record marker (**Peor**). In general, PADS provides a rich collection of array-termination conditions: reaching a maximum size, finding a terminating literal (including end-of-record and end-of-source), or satisfying a user-supplied predicate over the already-parsed portion of the **Parray**. Finally, this type declaration includes a **Pwhere** clause to specify that the sequence of timestamps must be in sorted order. It uses the **Pforall** construct to express this constraint. In general, the body of a **Pwhere** clause can be any boolean expression. In such a context for arrays, the pseudo-variable `elts` is bound to the in-memory representation of the sequence and `length` to its length.

Returning to the CLF description in Figure 2.3, the **Penum** type `method_t` describes a collection of

```
Punion client_t {
  Pip       ip;      /- 135.207.23.32
  Phostname host;    /- www.research.att.com
};

Punion auth_id_t {
  Pchar unauthorized : unauthorized == '-';
  Pstring(:' ':) id;
};

Penum method_t {
    GET, PUT, POST, HEAD, DELETE, LINK, UNLINK
};

Pstruct version_t {
  "HTTP/"; Puint8 major;
  '.';     Puint8 minor;
};

bool chkVersion(version_t v, method_t m) {
  if ((v.major == 1) && (v.minor == 1)) return true;
  if ((m == LINK) || (m == UNLINK)) return false;
  return true;
};

Pstruct request_t {
  '\"';   method_t       meth;
  ' ';    Pstring(:' ':) req_uri;
  ' ';    version_t      version :  chkVersion(version, meth);
  '\"';
};

Ptypedef Puint16_FW(:3:) response_t :
         response_t x => { 100 <= x && x < 600};

Precord Pstruct entry_t {
         client_t       client;
   ' ';  auth_id_t      remoteID;
   ' ';  auth_id_t      auth;
   " ["; Pdate(:']':)   date;
   "] "; request_t      request;
   ' ';  response_t     response;
   ' ';  Puint32        length;
};

Psource Parray clt_t {
  entry_t [];
}
```

Figure 2.3: PADS description for web server log data.

```
Precord Pstruct summary_header_t {
  "0|";     Puint32 tstamp;
};

Pstruct no_ramp_t {
  "no_ii";  Puint64 id;
};

Punion dib_ramp_t {
  Pint64     ramp;
  no_ramp_t  genRamp;
};

Pstruct order_header_t {
       Puint32             order_num;
 '|';  Puint32             att_order_num;
 '|';  Puint32             ord_version;
 '|';  Popt pn_t           service_tn;
 '|';  Popt pn_t           billing_tn;
 '|';  Popt pn_t           nlp_service_tn;
 '|';  Popt pn_t           nlp_billing_tn;
 '|';  Popt Pzip           zip_code;
 '|';  dib_ramp_t          ramp;
 '|';  Pstring(:'|':)      order_type;
 '|';  Puint32             order_details;
 '|';  Pstring(:'|':)      unused;
 '|';  Pstring(:'|':)      stream;
 '|';
};

Pstruct event_t {
  Pstring(:'|':) state;   '|';
  Puint32        tstamp;
};

Parray eventSeq {
  event_t[] : Psep('|') && Pterm(Peor) ;
} Pwhere {
  Pforall (i Pin [0..length-2] : (elts[i].tstamp <= elts[i+1].tstamp));
};

Precord Pstruct entry_t {
  order_header_t  header;
  eventSeq        events;
};

Parray entries_t {  entry_t[]; };

Psource Pstruct out_sum{
  summary_header_t  h;
  entries_t         es;
};
```

Figure 2.4: Partial PADS description for Sirius provisioning data.

data literals. During parsing, PADS interprets these constants using the ambient character encoding. The **Ptypedef** response_t describes possible server response codes in CLF data by adding the constraint that the three-digit integer must be between 100 and 600.

The order_header_t type in the Sirius data description contains several anonymous uses of the **Popt** type. This type is syntactic sugar for a stylized use of a **Punion** with two branches: the first with the indicated type, and the second with the "void" type, which always matches but never consumes any input.

2.3 Generated library

From a description, the PADS compiler generates a C library for parsing and manipulating the associated data source. To give a feeling for the library that PADS generates, Figure 2.5 includes selected portions of the generated library for the Sirius entry_t declaration.

2.3.1 Example library use

Figure 2.6 shows a simple use of the generated Sirius library to filter out ill-formed records and normalize the representation of optional phone numbers. The code first initializes the PADS library handle, p, by invoking the core library function P_open. The second argument to this function allows the user to customize behavior in the PADS library by passing a (pointer to a) Pdisc_t *discipline*. With this discipline, the user can specify properties such as the endianness of the data, the default character set (ASCII or EBCIDC), error handling, *etc.* The third argument to the P_open function specifies a (pointer to a) Pio_disc_t discipline, which allows the user to describe how to detect record boundaries, *i.e.*, are records new-line terminated, fixed width, EBCDIC-style, *etc.* Passing zero in these argument positions triggers default behavior, which means ASCII encoded, little endian, new-line terminated records. Chapter 15 describes the use of disciplines in more detail.

After initializing the PADS handle, the code uses the core library function P_io_fopen to open the data source by specify the path to the data on disk. The code then uses generated functions to initialize the mask, parse descriptor, and in-memory representations for the header and the entry types. The specified mask values instruct the parsing code to check all conditions in the Sirius description except the sorting of the timestamps. The P_Set flag instead instructs the compiler to simply set the in-memory representation for the timestamp sequence.

After reading the header, the code echoes error records to one file and cleaned ones to another. The raw data has two different representations of unavailable phone numbers: simply omitting the number altogether, which corresponds to the NONE branch of the **Popt**, or having the value 0 in the data. The function cnvPhoneNumbers unifies these two representations by converting the zeroes into NONEs. The function entry_t_verify ensures that our computation hasn't broken any of the semantic properties of the in-memory representation of the data.

Finally, the core library functions P_io_close and P_close close the IO stream and PADS library, respectively.

2.3.2 Accumulators

Before using a data source, analysts must develop an understanding of both the layout and the meaning of the data. Because documentation is usually incomplete or out-of-date, this understanding must be developed through exploring the data itself. Typical questions include: how complete is the description of the syntax of the data source, how many different representations for "data not available" are there, what is the distribution of values for particular fields, *etc.* PADS addresses these kinds of questions with the notion of an accumulator. For each type in a PADS description, accumulators track the number of good values, the number of bad values,

```
typedef struct {
  Pbase_m compoundLevel;   /* Struct-level controls, eg., check Pwhere clause */
  order_header_t_m h;
  eventSeq_t_m events;
} entry_t_m;

typedef struct {
  Pflags_t pstate;         /* Normal, Partial, or Panicking       */
  Puint32 nerr;            /* Number of detected errors.          */
  PerrCode_t errCode;      /* Error code of first detected error  */
  Ploc_t loc;              /* Location of first error             */
  order_header_t_pd h;     /* Nested header information           */
  eventSeq_t_pd events;    /* Nested event sequence information   */
} entry_t_pd;

typedef struct {
  order_header_t h;
  eventSeq_t events;
} entry_t;

/* Core parsing library */
Perror_t entry_t_read (P_t *pads,entry_t_m *m,entry_t_pd *pd,
                       entry_t *rep);

/* Selected utility functions */
void entry_t_m_init (P_t *pads,entry_t_m *mask,Pbase_m baseMask);
int entry_t_verify (entry_t *rep);

/* Selected accumulator functions */
Perror_t entry_t_acc_init (P_t *pads,entry_t_acc *acc);
Perror_t entry_t_acc_add (P_t *pads,entry_t_acc *acc,
                          entry_t_pd *pd,entry_t *rep);
Perror_t entry_t_acc_report (P_t *pads,char const *prefix,
                             char const *what,
                             int nst,entry_t_acc *acc);
ssize_t entry_t_write2io (P_t *pads,Sfio_t *io,entry_t_pd *pd,entry_t *rep);

/* Formatting */
ssize_t entry_t_fmt2io (P_t *pads,Sfio_t *io,int *requestedOut,
                        char const *delims,entry_t_m *m,entry_t_pd *pd,
                        entry_t *rep);

/* Conversion to XML */
ssize_t entry_t_write_xml_2io (P_t *pads,Sfio_t *io,entry_t_pd *pd,
                               entry_t *rep,char const *tag,int indent);
```

Figure 2.5: Selected portions of the library generated for the `entry_t` declaration from Sirius data description.

```
P_t                   *p;
summary_header_t      header;
summary_header_t_pd   header_pd;
summary_header_t_m    header_m;
entry_t               entry;
entry_t_pd            pd;
entry_t_m             mask;

P_open(&p, 0, 0);
P_io_fopen(p, "data/sirius");

/* Initialize header data structures */
summary_header_t_init(p,   &header);
summary_header_t_pd_init(p,&header_pd);
summary_header_t_m_init(p, &header_m, P_CheckAndSet);

/* Initialize entry data structures */
entry_t_init(p,    &entry);
entry_t_pd_init(p, &pd);
entry_t_m_init(p,  &mask, P_CheckAndSet);
mask.events.compoundLevel = P_Set;

/* Try to read header   */
if (P_OK == summary_header_t_read(p, &header_m, &header_pd, &header)) {
  summary_header_t_write2io(p, CLEAN_FILE, &header_pd, &header);
} else {
  error(2, "reading header returned: error");
}

/* Try to read each line of data  */
while (!P_io_at_eof(p)) {
  entry_t_read(p, &mask, &pd, &entry);
  if (pd.nerr > 0) {
    entry_t_write2io(p, ERR_FILE, &pd, &entry);
  } else {
    cnvPhoneNumbers(&entry);
    if (entry_t_verify(&entry)) {
      entry_t_write2io(p, CLEAN_FILE, &pd, &entry);
    } else {
      error(2, "Data transform failed.");
    }
  }
}
P_io_close(p);
P_close(p);
```

Figure 2.6: Fragment of a program to filter and normalize Sirius data.

```
#define DEF_INPUT_FILE "data/wsl"    /* Default data location             */
#define PADS_TY(suf) entry_t ## suf  /* Name of record type = entry_t     */
#define IO_DISC_MK P_nlrec_make(0)   /* Records are new line terminated   */
#include "wsl.h"                     /* Header file for generated library */
#include "template/accum_report.h"   /* Accumulator template program      */
```

Figure 2.7: Accumulator program to construct statistical profiles of web server log data.

and the distribution of legal values. Selected functions from this portion of the library appear in Figure 2.5; more detailed information appears in Chapter 16.

We can of course use these functions by hand to write a program to compute the statistical profile of any PADS data source. However, ad hoc sources are often simply a sequence of records, perhaps prefixed by a header. For example, both the web server log and the Sirius data sources exhibit this pattern, as does any data format that can be read in one bulk read. As a convenience, PADS supplies a program template to construct accumulator programs for such data sources. PADS provides this template as a C include file and supports customization via C's macro system. Using this template, we can write an accumulator program by specifying only the names of the optional header type and the record type. Figure 2.7 contains the entirety of the user-specified accumulator program text.

The accumulator report for the length field of the web server data that results when run on a data set used in several studies of web traffic [KW00, KW02] appears in Figure 2.8. By default, accumulators track the first 1000 distinct values seen in the data source and report the frequency of the top ten values. In this particular run, 99.552% of all values were tracked. When generating the accumulator program (or when using the library directly), PADS users can specify the number of distinct values to track and the number of values to print in the report. Details about customizing accumulator programs appear in Chapter 16.

Perhaps surprisingly, the report shows that 6.66% of the length fields contained errors. A glance at the error log generated by the program (which contains all records flagged as errors) reveals that web servers occasionally store the '-' character rather than the actual number of bytes returned, a possibility not mentioned in the documentation [KR01]. Accumulators often serve as a quick tool for iteratively refining a PADS description until only genuine errors remain.

2.3.3 Formatting

To support converting ad hoc data into a delimited format, the PADS library generates a formatting function for each type. This function, an example of which appears in Figure 2.5, takes a delimiter list as an argument. At each field boundary, it prints the first delimiter. At each nested type boundary, it advances the delimiter list unless the list is exhausted, in which case it reuses the last delimiter. The mask argument allows the user to suppress printing of portions of the data. Programmers can use the library directly to write formatting programs by hand. However, as in the accumulator case, PADS can generate a formatting program for commonly occurring data patterns given only the header type (optional), record type, and a delimiter string. Users can further customize the generated program by specifying an output format for dates and mask values as illustrated in Figure 2.9. Given the delimiter string `"|"` and the output date format `"%D:%T"`, the generated web server log formatting program yields the output shown in Figure 2.10 when applied to the sample data in Figure 2.1.

To support customization, PADS allows users to provide their own formatting functions for any type. More information on formatting functions can be found in Chapter 19.

```
<top>.length : uint32
++++++++++++++++++++++++++++++++++++++++
good: 53544   bad: 3824    pcnt-bad: 6.666
min: 35  max: 248591  avg: 4090.234
top 10 values out of 1000 distinct values:
tracked 99.552% of values
 val:  3082 count:  1254  %-of-good:  2.342
 val:   170 count:  1148  %-of-good:  2.144
 val:    43 count:  1018  %-of-good:  1.901
 val:  9372 count:   975  %-of-good:  1.821
 val:  1425 count:   896  %-of-good:  1.673
 val:   518 count:   893  %-of-good:  1.668
 val:  1082 count:   881  %-of-good:  1.645
 val:  1367 count:   874  %-of-good:  1.632
 val:  1027 count:   859  %-of-good:  1.604
 val:  1277 count:   857  %-of-good:  1.601
. . . . . . . . . . . . . . . . . . . . .
 SUMMING    count:  9655  %-of-good: 18.032
```

Figure 2.8: Portion of accumulator report for length field of web server log data.

2.3.4 Conversion to XML

PADS also supports converting ad hoc data into XML by providing a canonical mapping from PADS descriptions into XML. This mapping is quite natural, as both PADS and XML are languages for describing semi-structured data. One interesting aspect of the mapping is that we embed not just the in-memory representation of PADS values, but also the parse descriptors in cases where the data was buggy. This choice allows users to explore the error portions of their data sources, which can be the most interesting parts of the data. Given a PADS specification, the PADS compiler generates an XML Schema describing the canonical embedding for that data source. As an example, Figure 2.11 shows the portion of the XML Schema generated from the Sirius data description related to the `eventSeq` type.

The PADS compiler will put the generated schema in the same directory as the generated library. The schema file will share the name of the PADS description and have the extension `xsd`.

The PADS compiler generates a `write_xml_2io` function for each type, an example of which is shown in Figure 2.5. Programmers can use these functions by hand to write conversion programs, or they can use

PADS also provides a template for generating such conversion programs for data that can be read entirely into memory.[1] Figure 2.12 contains an example of such a program. More information about converting to XML can be found in Chapter 20.

[1]The XML template program also permits the programmer to specify a header type followed by a body record type. However, the generated XML does not include the appropriate top-level declarations to conform to the generated XML schema. We are working on addressing this problem.

```
#define DEF_INPUT_FILE "data/wsl"
#include "wsl.h"
#define PADS_TY(suf) entry_t ## suf
#define IO_DISC_MK P_nlrec_make(0)
#define DATE_OUT_FMT "%D:%T"
#define DELIMS "|"
#include "template/read_format.h"
```

Figure 2.9: PADS program for formatting CLF records using formatting template.

```
207.136.97.49|-|-|10/16/97:01:46:51|GET|/tk/p.txt|1|0|200|30
tj62.aol.com|-|-|10/16/97:21:32:22|POST|/scpt/dd@grp.org/confirm|1|0|200|941
```

Figure 2.10: Formatted CLF records.

```
<xs:complexType name="eventSeq_pd">
<xs:sequence>
<xs:element name="pstate" type="Pflags_t"/>
<xs:element name="nerr" type="Puint32"/>
<xs:element name="errCode" type="PerrCode_t"/>
<xs:element name="loc" type="Ploc_t"/>
<xs:element name="neerr" type="Puint32"/>
<xs:element name="firstError" type="Puint32"/>
<xs:element name="elt" type="Puint32"
    minOccurs="0" maxOccurs="unbounded"/>
</xs:sequence>
</xs:complexType>

<xs:complexType name="eventSeq">
<xs:sequence>
<xs:element name="elt" type="event"
    minOccurs="0" maxOccurs="unbounded"/>
<xs:element name="length" type="Puint32"/>
<xs:element name="pd" type="eventSeq_pd"
    minOccurs="0" maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
```

Figure 2.11: Portion of XML Schema generated from Sirius data description.

```
#define DEF_INPUT_FILE  "data/sirius"       /* Default data source             */
#define PADS_TY(suf) out_sum ## suf         /* Record type = out_sum           */
#define IO_DISC_MK P_nlrec_make(0)          /* Records are new line terminated */
#include "sirius.h"                         /* Header file for data source     */
#include "template/read_orig_write_xml.h"   /* XML Conversion template         */
```

Figure 2.12: Program to convert Sirius data into XML using PADS template.

Chapter 3

Common features

In this chapter, we describe the PADS features shared by all types. Subsequent chapters describe features particular to individual PADS types.

3.1 PADS types

Each PADS type specifies the external representation of a particular kind of data. PADS base types describe the representations of atomic pieces of data, while structured types specify how compound representations are built from more basic ones. PADS provides a large and extensible collection of base types and a family of type constructors for building structured types: **`Pstruct`**s for record-like sequences, **`Punion`**s for alternatives, **`Parray`**s for sequences, **`Penum`**s for fixed collections of strings, and **`Ptypedef`**s for refinements of existing types.

Syntactically, a PADS type declaration (*p_ty_decl*) must have the following form:

p_ty_decl	::=	*base_ty*	(* Chapter 4 *)
	|	*struct_ty*	(* Chapter 5 *)
	|	*union_ty*	(* Chapter 6 *)
	|	*array_ty*	(* Chapter 7*)
	|	*enum_ty*	(* Chapter 8 *)
	|	*opt_ty*	(* Chapter 9 *)
	|	*typedef_ty*	(* Chapter 10 *)
	|	*trans_ty*	(* Chapter 11 *)
	|	*try_ty*	(* Chapter 12 *)
	|	*charclass_ty*	(* Section 3.14.1 *)

We use the terminal *p_ty_name* for an identifier bound to one of the above types.

The grammar for each of the above non-terminals is given in the associated chapter.

3.2 Comments

In addition to C and C++ style comments of the form `/* ... */` and `//...`, which may appear anywhere in a PADS description, PADS also supports *p_comment*s, which may appear only in particular locations in the grammar. Syntactically,

p_comment ::= */- text*

where *text* is a new-line terminated sequence of characters. PADS comments are reflected to the generated `.h` file as documentation. We indicate where such comments may appear in the source as the locations arise in the descriptions of the various PADS features.

3.3 Predicates

PADS descriptions permit the user to supply *predicates* for validating semantic properties of syntactically correct data. Syntactically, such predicates are arbitrary C expressions of integer type. Predicates that evaluate to `false` (*i.e.*, `0`) imply the data is invalid, while all other values imply the data is valid. Predicates are assumed to be side-effect free.

To allow users to express constraints involving the size and position of physical data, PADS supports *p_parsecheck* expressions within predicates. Syntactically,

p_parsecheck ::= **Pparsecheck**(*aug_expression*)

In the production, *aug_expression* is an integer-valued C expression that is allowed to refer to special constants providing location information. The precise constants depend upon the context of the **Pparsecheck** clause in the PADS description, but always include the constant `position` of type `Ppos_t`, bound to the current position in the physical source.

For integration with general predicates, **Pparsecheck** expressions are treated as C expressions with type `int`.

3.4 Literals

PADS supports C-style character and string literals, referred to in the PADS grammar as *char_lit* and *str_lit*, respectively. These literals may contain C character escapes such as `\"` and `\'`. PADS also supports regular expression literals, described in more detail in Section 3.14, and the special literal **Peor**, which denotes the end of a record. We use the non-terminal *regexp_lit* to refer to regular expressions. Syntactically,

p_coreliteral ::= *char_lit* | *str_lit* | **Pre** *regexp_lit* | **Peor**

p_literal ::= *p_coreliteral* | *C_identifer* **Pfrom** (*p_coreliteral*)

Literals (as opposed to core literals) also support a renaming form. The supplied C identifier gives the programmatic name for the literal, while the core literal supplied in the **Pfrom** clause describes the on-disk representation. Renaming can useful when the literal is not a valid C identifier.

3.5 Character Sets

The library discipline contains a field `def_charset` that indicates the expected character set of the external representation of character and string literals, as well as the external representation of all character and string base types that do not explicitly name a character set. Supported character sets include ASCII (`Pcharset_ASCII`) and EBCDIC (`Pcharset_EBCDIC`), where ASCII is the default character set. Section 15.1.3 describes how to set `def_charset`.

3.6 Parameterization

To reduce the number of necessary type declarations and to permit the format of later portions of the data to depend upon earlier portions, PADS types can be parameterized by values. A common example of the latter use is a data source which first specifies the length of a sequence and then gives the sequence itself. The length is read, stored in the in-memory representation, and then passed to the type that describes the sequence to specify the termination condition for the sequence. Syntactically,

p_actual_list ::= *expression* | *expression*, *p_actual_list*
p_actuals ::= (: *p_actual_list* :)
p_ty ::= p_ty_name [*p_actuals*]

where *expression* is any C expression. Formal parameter lists are similar to C's, except they are deliminated differently:

p_formals ::= (: *c_formal_list* :)

3.6.1 Example

The `formalParamExample` type declaration in the following PADS fragment illustrates declaring parameters to PADS types, while the `actualParamExample` illustrates passing parameters.

```
Pstruct formalParamExample(:Pint32 limit:){
  Pint32 data  :  data <= limit;
}

Pstruct actualParamExample{
  Pint32 limit;
  formalParamExample(:limit:) nestedData;
}
```

Type `formalParamExample` expects its single field `data` to be less than a supplied value `limit`, while the type `actualParamExample` describes an external representation with two pieces: an integer `limit` and then an instance of `formalParamExample`. The value of `limit` is passed as the actual parameter to the `formalParamExample` **`Pstruct`**.

3.7 `Precord` modifier

The **`Precord`** modifier may be used as an annotation on any PADS type, indicating that the type describes a *record* in the external representation of the data. PADS supports a number of different interpretations of what

constitutes a record:

Record type	Description
New-line terminated	End of record marked by new-line character
Fixed-width record	Records contain a specified number of bytes
IBM-style record	Record header indicates size

Section 15.2 describes how to set the appropriate record discipline for a data source.

3.8 `Psource` modifier

The **`Psource`** modifier may be used as an annotation on any PADS type, indicating that the type in question describes the entirety of the external representation of the data.

3.9 `Pinclude`

The **`Pinclude`** statement within a `.p` file causes the supplied argument to be mirrored to the generated `.h` file, but not the generated `.c` file. It is useful for including necessary header files for C code referenced in the PADS description. For example, the following use causes the generated `.h` file to include the directive `#include <rpc/rpc.h>`.

```
Pinclude(:#include <rpc/rpc.h>:)
```

3.10 Error model

During parsing, PADS read functions detect when the data does not conform to the given specification. Detected violations fall into two classes, which differ in their severity. The less severe of these, *semantic* errors are those in which the parser detects a violation of the specified format but does not "lose its place." A typical example of this kind of error is a violation of a user-specified constraint, such as a requirement that a given field be greater than a threshold value. The more severe type of error, *syntactic* errors, involve the parser finding raw data that cannot be reconciled with the physical aspects of the description. Typical examples include failing to find literals required by **`Pstruct`** declarations or separators from **`Parray`** declarations. We say that a read function enters *panic* mode when it detects such an error. Read functions set the `P_Panic` flag in the `pstate` field of the appropriate parse descriptor when entering panic mode (Section 3.13 describes the general role and structure of parse descriptors).

PADS read functions attempt to recover from panic mode by scanning for possible synchronization points in the data source. For example, if the read function for a PADS type `foo` annotated with the **`Precord`** qualifier enters panic mode, it tries to recover by scanning to the end of the record. If it succeeds in finding the record boundary, it lowers the panic flag in the `foo` parse descriptor, although the nested parse descriptor for the portion of the data description that caused the panic will still be set. The parse descriptions for all portions of the data description that were skipped during the scanning process will also have the panic flag set. Type-specific information regarding error-recovery appears in the corresponding chapter.

3.11 In-memory representations

Each PADS type `foo` has an associated in-memory representation type of the same name. The structure of this representation depends upon the particular PADS type. In general, these representations fall into two broad categories: *static* representations, whose size can be computed at library-generation time, and *dynamic* representations, whose size depends on the data being parsed. Details appear in Section 4.1, Section 5.2.1, Section 6.2.2, Section 7.3.1, Section 8.2.1, Section 10.2.1, and Section 11.2.1.

3.12 Masks

Each PADS type `foo` has an associated mask type, `foo_m`. Masks allow the library user to customize operations on portions of the associated data. The structure of the mask for a given PADS type mirrors the structure of the representation type. Details about the structure for various types appear in Section 4.2, Section 5.2.2, Section 6.2.3, Section 7.3.2, Section 8.2.2, Section 10.2.2, and Section 11.2.2. Different operations in the generated library interpret masks differently. Details about how a given operation treats its mask argument appear in the sections describing the operations.

3.13 Parse descriptors

Each PADS type `foo` has an associated parse descriptor type, `foo_pd`, coded as a C struct with at least the following four fields:

Field	Description
`Puint32 pstate`	Flags that describe the state after parsing the associated value.
`PerrCode_t errCode`	A code indicating the nature of the first detected error. Appendix A contains a list of all error codes and describes their meanings.
`Ploc_t loc`	The location in the data source of the first error.
`Puint32 nerr`	The number of errors detected during parsing of the associated value.

Field `pstate` contains the following flags:

`PPanic` Set if the parser was in panic mode during the parsing of the associated data. See Section 3.10 for more information.

The PADS library provides a collection of functions (macros actually) for manipulating the parse state field:

```
void P_PS_init(void *pd);
void P_PS_setPanic(void *pd);
void P_PS_unsetPanic(void *pd);
int  P_PS_isPanic(void *pd);
```

The `loc` field record the location of the related data in the source file. A `Ppos_t` (IO position) has a `byte` position within the `num`'th read unit, where the read unit is determined by the IO discipline. A description of the read unit (*e.g.*, `"record"`, `"1K Block"`, *etc.*) can be obtained using the function `P_io_read_unit` described in Chapter 14 There is also an `offset` field which gives the absolute offset of the location within the currently installed IO stream.

```
typedef struct Ppos_s {
  size_t       byte;
  size_t       num;
  Sfoff_t      offset;
} Ppos_t;
```

A `Ploc_t` (IO location) has two positions, `b` and `e`, marking the first byte and the last byte where something interesting happened, *e.g.*, a field with an invalid format.

```
struct Ploc_s {
  Ppos_t b;
  Ppos_t e;
};
```

In cases where clearcut boundaries for an error are not known, the parse position where the error was 'found' is used for both the begin and end positions. In this case, and in some other cases, the end byte is set to one less than the start byte, indicating an error that occurred just before the start byte (as opposed to an error that spans the start byte).

The beginning location for a data item is always filled in. The ending location is only filled in if there is an error.

Details about how parse descriptors are customized for various PADS types appear in Section 4.3, Section 5.2.3, Section 6.2.4, Section 7.3.3, Section 8.2.3, Section 10.2.3 and Section 11.2.3.

3.14 Regular Expressions

p_regexp_lit ::= "/ `regexp` /"

p_regexp_expression ::= **`Pre`** *expression*

PADS regular expressions support the full posix regex specification `www.opengroup.org/onlinepubs/009695399/`, and also support many of the Perl extensions. If you have Perl installed, you can use

```
> man perlre
```

to see Perl's regular expression man page.

A regular expression is specified in a PADS description as a string (a **`const char`**$\star$). The first character in the string is the expression delimiter: the next (non-escaped) occurrence of this delimiter marks the end of the regular expression. We typically write our examples using slash (`/`) as the delimiter, but any delimiter can be used. After the closing delimiter, one can add one or more single-character modifiers which change the normal matching behavior. The modifiers are based on those supported by Perl, and currently include:

Modifier	Meaning
l	Treat the pattern as a literal. All characters in the pattern are literal characters to be found in the input. There are no operators or special characters.
i	Do case-insensitve pattern matching.
x	Extend your pattern's legibility by permitting whitespace and comments. Tells the regular expression parser to ignore whitespace that is neither backslashed nor within a character class You can use this to break up your regular expression into (slightly) more readable parts. The `"#"` character is also treated as a metacharacter introducing a comment. This also means that if you want real whitespace or `"#"` characters in the pattern (outside a character class, where they are unaffected by the `x` modifier), you'll either have to escape them or encode them using octal or hex escapes. Be careful not to include the pattern delimiter in the comment – there is no way of knowing you did not intend to close the pattern early.
?	Minimal match. Change from the normal maximal left-most match semantics to a minimal left-most match semantics.
f	First match. Change from the normal maximal left-most match semantics to accepting the first match found. This may be useful for terminating regular expressions where any match is sufficient to trigger termination. For termination, the matched characters are not included in the resulting value, so getting the best set of matching characters may not be necessary.

It is important to note that in normal POSIX regexps, the `'$'` and `'^'` special characters match "beginning of line" and "end of line" respectively, where newline is the line separator character. In contrast, in PADS regular expressions, the `'$'` and `'^'` special characters match "beginning of record" and "end of record" respectively (and thus they only have meaning with the record-based IO disciplines). For this reason, newlines that occur within records or within input data for non-record-based input are treated as normal characters with no special semantics. This means, for example, that the `'.'` special character will match newlines. (In Perl one would use the `"/s"` modifier to get similar behavior.)

If newlines in your input data mark record boundaries, you should be using one of the `nlrec` IO disciplines described in Section 15.2, in which case the newlines do not appear in your normal input, so there is no issue of `'.'` matching newlines, and `'$'` and `'^'` will have their normal POSIX behavior.

A regular expression that uses *both* `'^'` and `'$'` may have problems matching arbitrarily large strings because the implementation divides the input into chunks of a particular size for processing. If the records in a data source are larger than this size, the regular expression will not have access to the entire record for matching. In more detail, the matching code finds a region [`begin`, `end`] to match over and determines whether `begin` is actually the beginning of the record and whether `end` is actually the end of the record. If `begin` is not really the beginning of the record, then it disables `'^'`, and if `end` is not really the end of the record, then it disables `'$'`. This implementation choice will effectively prevent a successful match if the regular expression uses `'^'` or `'$'` unless the regular expression has alternation with a clause that does not use the disabled `'^'` or `'$'`. The scope of matching is controlled by the PADS discipline, as described in Section 15.1.5.

Regular expressions are used for two purposes in PADS, and the matching semantics with respect the current IO position are different for these two cases, as follows.

- A regular expression can be used as the inclusive scope of a data field, *i.e.*, it defines the set of characters that will be included in a resulting value (see `Pstring_ME` / `Pstring_CME`).

  In this case, the regular expression is implicitly left-bounded at the current IO position: if a match cannot be found that includes the character at the current IO position, then matching fails.

  The default is that the longest such match will be used that is within the scope determined by `pads->disc->match_max`, described in Section 15.1.5

- A regular expression can be used to terminate a data field (see `Pstring_SE` / `Pstring_CSE`).

In this case, the regular expression is not "left bounded": the matcher finds the longest match whose first-¿last characters occur anywhere in the scope determined by `pads->disc->scan_max`.

The resulting value consists of all characters from the current IO position up to (but not including) the left-most character in the match, *i.e.*., none of the characters in the match are included in the value; the match simply *terminates* the value.

Example: suppose a string is either terminated by a comma or by end-of-record. This would by specified in a PADS description as:

```
Pstring_SE(:"/[,]|$/":)     my_string;
```

Within regular expressions, one can write in brackets `[]` a set of characters to be matched against, or the inverse of such a set:

`[abc]`	matches an `'a'`, `'b'`, or `'c'`
`[^abc]`	matches any character EXCEPT an `'a'`, `'b'`, or `'c'`

INSIDE of one of these bracket expressions one can include a character class using the syntax `[:<classname>:]`. For example, the following matches either a letter (`'A'` through `'Z'` or `'a'` through `'z'`) or a `'0'` or `'1'`:

```
[0[:alpha:]1]
```

Using character classes is preferable to writing something like this:

```
[0A-Za-z1]
```

because the letters `A-Z` may not occur contiguosly in all character set encodings. Note that when you just specify a character class within brackets, you end up with a double set of brackets, as in this pattern representing one more alpha characters:

```
/[[:alpha:]]+/
```

The following are all built-in character classes:

`[:alnum:]`	alpha or digit
`[:alpha:]`	upper or lower alphabet character
`[:blank:]`	space (' ') or tab
`[:cntrl:]`	control character
`[:digit:]`	digit (0 through 9)
`[:graph:]`	any printable character except space
`[:lower:]`	lower-case letter
`[:print:]`	any printable character including space
`[:punct:]`	any printable character which is not a space or an alphanumeric character
`[:space:]`	a white-space character. Normally this includes: space, form-feed (`'\f'`), newline (`'\n'`), carriage return (`'\r'`), horizontal tab (`'\t'`), and vertical tab (`'\v'`)
`[:upper:]`	an upper-case letter
`[:word:]`	an alphanumeric character or an underscore (`'_'`)
`[:xdigit:]`	a hexadecimal digit (normal digits and `A` through `F`)

3.14.1 Defining your own character classes

It is possible to define your own character class in a PADS file and then use that class in regular expressions that occur later in the file.

Syntax

charclass_ty ::= **Pcharclass** identifier { identifier};

In this grammar, the first identifier names the character class, while the second names a predicate function which takes a **char** as an argument and returns an **int** as a result. Intuitively, this function returns a non-zero value to indicate that the argument character belongs to the class and a zero to indicate it does not.

For example, the following code defines the `foo` character class:

```
int is_foo(char c) { return (c == 'f') || (c == 'o') || isdigit((int)c); };

Pcharclass foo {is_foo};
```

Section 14.1 describes compiling regular expressions.

3.15 Expressions

Expression forms include C expressions, regular expressions, and the special symbol **Peor**.

p_expression ::= *expression* | **Pre** *expression* | **Peor**

3.16 Operations

For each PADS type, the generated library contains a collection of functions for manipulating the associated data. For structured types, the PADS compiler generates the functions; for base types, the PADS library provides them. This section describes the common features of such functions. Type-specific information may be found in the appropriate chapter.

All operations take a pointer to an initialized PADS handle as their first parameter. Information about how to manage PADS library handles appears in Chapter 14.

Many of the operations that can fail return a value of type `Perror_t` to indicate success or failure. This type has two values: `P_OK` and `P_ERR`.

3.16.1 Initialization and cleanup functions

For each type `foo`, the generated library contains initialization and cleanup functions for the associated representation `foo` and parse descriptor `foo_pd` types. Each initialization function take a pointer to allocated space and initializes the space appropriately. Each cleanup function takes a pointer to allocated and initialized space and deallocates any memory allocated by PADS functions. It does not deallocate the space pointed to by the parameter.

```
Perror_t foo_init (P_t *pads, foo *rep);

Perror_t foo_cleanup (P_t *pads, foo *rep);

Perror_t foo_pd_init (P_t *pads, foo_pd *pd);

Perror_t foo_pd_cleanup (P_t *pads, foo_pd *pd);
```

Because all masks have statically-known size, the library does not contain initialization and cleanup functions for masks. Instead, it contains a function for setting all nested base masks to a specified value:

```
void foo_m_init (P_t *pads, foo_m *mask, Pbase_m baseMask);
```

The function takes two parameters (in addition to the PADS handle): a pointer to allocated space for the mask and a base mask value. Because mask initialization functions cannot fail, the return type is `void` instead of `Perror_t`.

3.16.2 Utilty functions

Each type `foo` comes equipped with copy functions for both the in-memory representation and the parse descriptor. Both the source and destination pointers are assumed to point to allocated space. In addition, the source pointers are assumed to point to initialized space.

```
Perror_t foo_copy (P_t *pads, foo *rep_dst, foo *rep_src);

Perror_t foo_pd_copy (P_t *pads, foo_pd *pd_dst, foo_pd *pd_src);
```

Each type `foo` also has a predicate function that returns true if the supplied in-memory representation satisfies all of the non-**`Pparsecheck`** constraints.

```
int foo_verify(P_t *pads, foo *rep);
```

In addition, each structured type `foo` has a function `foo_genPD` that takes as an argument an in-memory representation and calculates a corresponding parse descriptor.

```
int foo_genPD (P_t *pads, foo *rep, foo_pd *pd);
```

The function assumes that the out parameter `pd` has already been allocated and initialized to zero. It sets the error codes and counts based on the semantic predicates found in the description of `foo`. The location information in the argument `pd` will not be touched. Future versions of the parse descriptor generation function may compute on-disk sizes. The function returns true if the representation contains no errors and false otherwise. The function takes the `pads` handle as an argument because it must allocate space for array parse descriptors.

3.16.3 Read function

The read function for a PADS type `foo` takes as an input parameter a pointer to a mask `m` and returns as output parameters a pointer to a parse descriptor `pd` and a pointer to an in-memory reprsentation `rep`. If any errors occur during the parsing, the function will return `P_ERR`. Otherwise, it will return `P_OK`.

```
Perror_t foo_read (P_t *pads, foo_m *m, foo_pd *pd, foo *rep);
```

The mask argument allows the library user to specify independently which constraints the parser should check and which portions of the in-memory representation it should fill in. Conceptually, there are three different "knobs" associated with each atomic element of a PADS type:

Flag name	Definition
`P_SynCheck`	Check syntactic constraints
`P_SymCheck`	Check semantic constraints
`P_Set`	Set the in-memory representation

At the base-type level, the mask consists of one or more of the above flags. For structured types, the mask consists of a combination of base masks and the masks associated with nested types, the exact combination of which depends upon the kind of structured type. Note that for a structured type `foo`, the mask initialization function `foo_m_init` can be used to initialize all nested masks to the supplied value.

```
#define P_Test_Set(m)             (m & P_Set)
#define P_Test_SynCheck(m)        (m & P_SynCheck)
#define P_Test_SemCheck(m)        (m & P_SemCheck)

#define P_Test_NotSet(m)          (!P_Test_Set(m))
#define P_Test_NotSynCheck(m)     (!P_Test_SynCheck(m))
#define P_Test_NotSemCheck(m)     (!P_Test_SemCheck(m))

#define P_Test_CheckAndSet(m)     ((m & P_CheckAndSet) == P_CheckAndSet)
#define P_Test_BothCheck(m)       ((m & P_CheckAndSet) == P_BothCheck)
#define P_Test_Ignore(m)          ((m & P_CheckAndSet) == P_Ignore)

#define P_Test_NotCheckAndSet(m) ((m & P_CheckAndSet) != P_CheckAndSet)
#define P_Test_NotBothCheck(m)    ((m & P_CheckAndSet) != P_BothCheck)
#define P_Test_NotIgnore(m)       ((m & P_CheckAndSet) != P_Ignore)

#define P_Do_Set(m)               (m |= P_Set)
#define P_Do_SynCheck(m)          (m |= P_SynCheck)
#define P_Do_SemCheck(m)          (m |= P_SemCheck)

#define P_Dont_Set(m)             (m &= (~ P_Set))
#define P_Dont_SynCheck(m)        (m &= (~P_SynCheck))
#define P_Dont_SemCheck(m)        (m &= (~P_SemCheck))
```

Figure 3.1: Provided macros for setting and testing base read-function flags.

Details about the read functions for particular types may be found in the appropriate chapters.

Flags can be combined using the bit-wise OR operator (vertical bar, or '`|`'). For example, one can write `P_Set|P_SynCheck|P_SemCheck` to combine the first three flags. For convenience, `pads.h` provides the following abbreviations:

`#define P_CheckAndSet`	`P_Set|P_SynCheck|P_SemCheck`
`#define P_BothCheck`	`P_SynCheck|P_SemCheck`
`#define P_Ignore`	no flags set (do as little work as possible to process this field)

The library also provides macros for testing and modifying base read-function flags, which are listed in Figure 3.1.

3.16.4 Write functions

For each PADS type, the generated library provides two functions for writing out the in-memory representation of the type in a format as close as possible to its original form.

```
ssize_t foo_write2io (P_t *pads, Sfio_t *io, foo_pd *pd, foo *rep);

ssize_t foo_write2buf (P_t *pads, Pbyte *buf, size_t buf_len,
                       int *buf_full, foo_pd *pd, foo *rep);
```

The first of these functions emits its output to an SFIO file; the second to an in-memory buffer. Information about SFIO may be found from `www.research.att.com/sw/tools/sfio`. For the buffer version, the parameter `buf` points to an allocated sequence of bytes and `buf_len` indicates the size of the buffer. Out parameter `buf_full` is a boolean which is set if the requested write would have overflowed the buffer. The return value for both of the functions is the number of bytes written, with `-1` indicating an error occurred.

Issues that may cause the written data to differ from the original data include skipping white space (Section 15.1.2) and omitting fields in **`Pstruct`**s (Section 5.1.6).

Passing the `-wnone` flag to the PADS compiler suppresses the generation of write functions for structured types.

3.16.5 Additional functions

In addition to the basic functionality described in the preceeding sections, PADS provides more advanced features described in later chapters.

Accumulators (Chapter 16)	Provide structures and functions for automatically summarizing data.
Histograms (Chapter 17)	Provide structures and functions for computing histograms.
Clustering (Chapter 18)	Provide structures and functions for clustering data.
Formatting (Chapter 19)	Provide functions for formatting data in forms suitable for inclusion in relational databases.
XML (Chapter 20)	Provide functions for converting data into canonical XML form as well as generating a corresponding XSCHEMA.

Chapter 4

Base Types Overview

PADS base types describe inidivual, small values: numbers, strings, dates, and so on. This chapter provides an overview of some of the most important built-in PADS base types, with a focus on how these types are used within PADS source files. Appendix B gives detailed descriptions of each of the built-in base types, including the full set of library API calls for each type. (As discussed in Chapter 3, each type has correspond read, write, format, and accumulator functions.)

In addition to the built-in types, it is possible to extend PADS with new base types; see Section 15.3.

4.1 In-Memory Representation

Each base type has an external and an in-memory representation. Related base types share the same in-memory representation. For example, while there are 18 different string base types, all of them use `Pstring` as their in-memory representation.

This section reviews the different in-memory representation types.

4.1.1 `Pchar`

Type `Pchar` is the in-memory representation of an external character. It is equivalent to the C type `unsigned char`, or type `Puint8`: all are 8-bit unsigned values. **N.B.**: Regardless of the external character that is read, the corresponding ASCII character is stored in the in-memory representation.

4.1.2 `Pstring`

Type `Pstring` is the in-memory representation for all forms of external strings. A `Pstring s` has two fields:

- `s.len` : the length of the string.
- `s.str` : a pointer to a sequence of `s.len` characters.

In addition, `Pstring` has fields that are manipulated by various string functions (some of which are described below). Most programmers should only use `s.len` and `s.str`.

```
Perror_t Pstring_init(P_t *pads, Pstring *s);
Perror_t Pstring_cleanup(P_t *pads, Pstring *s);
Perror_t Pstring_share(P_t *pads, Pstring *targ, const Pstring *src);
Perror_t Pstring_cstr_share(P_t *pads, Pstring *targ, const char *src, size_t len);
Perror_t Pstring_copy(P_t *pads, Pstring *targ, const Pstring *src);
Perror_t Pstring_cstr_copy(P_t *pads, Pstring *targ, const char *src, size_t len);
Perror_t Pstring_preserve(P_t *pads, Pstring *s);
int Pstring_eq(const Pstring *str1, const Pstring *str2);
int Pstring_eq_cstr(const Pstring *str, const char *cstr);

Pint8    Pstring2int8  (const Pstring *str);  /* returns P_MIN_INT8 on error   */
Pint16   Pstring2int16 (const Pstring *str);  /* returns P_MIN_INT16 on error  */
Pint32   Pstring2int32 (const Pstring *str);  /* returns P_MIN_INT32 on error  */
Pint64   Pstring2int64 (const Pstring *str);  /* returns P_MIN_INT64 on error  */

Puint8   Pstring2uint8 (const Pstring *str);  /* returns P_MAX_UINT8 on error  */
Puint16  Pstring2uint16(const Pstring *str);  /* returns P_MAX_UINT16 on error */
Puint32  Pstring2uint32(const Pstring *str);  /* returns P_MAX_UINT32 on error */
Puint64  Pstring2uint64(const Pstring *str);  /* returns P_MAX_UINT64 on error */

Pfloat32 Pstring2float32(const Pstring *str); /* returns P_MIN_FLOAT32 on error */
Pfloat64 Pstring2float64(const Pstring *str); /* returns P_MIN_FLOAT64 on error */
```

Figure 4.1: Library functions for manipulating `Pstring`s.

The library discipline has a field `copy_strings` which controls copying behavior for string read calls. If `copy_strings` is non-zero, the string read functions always copy strings. Otherwise, a copy is not made and the target `Pstring` points to memory managed by the current IO discipline. `copy_strings` should only be set to zero for record-based IO disciplines where strings from record K are not used after `P_io_next_rec` has been called to move the IO cursor to record K+1. Note: `Pstring_preserve` can be used to force a string that is using sharing to make a copy so that the string is 'preserved' (remains valid) across calls to `P_io_next_rec`.

When copying is used, the string copies are stored in an internal resizable buffer, and `s.str` points into this buffer. To ensure correct behavior, function `Pstring_init` should be called prior to using a `Pstring`, and function `Pstring_cleanup` should be called when a given `Pstring` is no longer in use. Generated initialization and cleanup functions call these routines for any PADS type containing a `Pstring`.

The full set of string helper functions appears in Figure 4.1.

The following list describes the behaviors of these functions.

`Pstring_init(pads, s)` Initialize `s` to valid empty string (no dynamic memory allocated yet).

`Pstring_cleanup(pads, s)` Free any dynamic memory allocated for `s`.

`Pstring_share(pads, targ, src)` Make `targ` refer to the string in `src`, sharing the space with the original owner.

`Pstring_cstr_share(pads, targ, src, len)` Make `targ` refer to `len` characters in C-string `src`.

`Pstring_copy(pads, targ, src)` Copy the string in `src` into `targ`; sharing is not used.

`Pstring_cstr_copy(pads, targ, src, len)` Copy `len` characters from C-string `src` into `targ`; sharing is not used.

```
P_MIN_INT8
P_MAX_INT8
P_MAX_UINT8

P_MIN_INT16
P_MAX_INT16
P_MAX_UINT16

P_MIN_INT32
P_MAX_INT32
P_MAX_UINT32

P_MIN_INT64
P_MAX_INT64
P_MAX_UINT64
```

Figure 4.2: Minimum and maximum values for PADS integer types.

`Pstring_eq(str1, str2)` Returns `1` if `str1` and `str2` are of equal length and `str1` equals `str2` (based on strncmp). Otherwise, returns `0`.

`Pstring_eq_cstr(str, cstr)` Returns `1` if `str1` and `str2` are of equal length and `str1` equals `str2` (based on strncmp). Otherwise, returns `0`.

Although not strictly necessary, both `Pstring_copy` and `Pstring_cstr_copy` null-terminate `targ->str`. Each copy function returns `P_ERR` on bad arguments or on failure to allocate space, otherwise it returns `P_OK`.

The various Pstring2NUMERIC functions convert a string to the specified numeric type. If the contents of the string cannot be converted to the specified type, the minumum value for the numeric type is returned for signed numeric types, or the maximum value is returned for unsigned numeric types.

4.1.3 Integer types

There are eight in-memory representations for integers, four types for signed values (`Pint8`, `Pint16`, `Pint32`, and `Pint64`) and four types for unsigned values (`Puint8`, `Puint16`, `Puint32`, and `Puint64`). The number in these type names indicates the number of bits in the in-memory representation, thus there are signed and unsigned integers that use 1, 2, 4, or 8 bytes of memory. The endian-ness of these in-memory representation types is the same as the endian-ness of the processor that the code is executing on. The external representation, which may be in another format, is always converted to the primitive in-memory representation supported by the processor.

For programming convenience, the header file `pads.h` includes definitions of the minimum and maximum values for each signed type, and of the maximum value for each unsigned type. Figure 4.2 list these constants.

4.1.4 Floating-point types

PADS has only two in-memory floating point representations, `Pfloat32` and `Pfloat64`, which correspond to ANSI C types `float` and `double`, respectively.

4.1.5 Fixed-point types

A fixed-point number is a number with a fixed number of decimal digits (digits after the 'dot'). For example, 123.456 is a fixed-point number with three decimal digits. External formats for such numbers occur, e.g., in COBOL data. We have chosen a very simple in-memory representation for such numbers: a struct with a numerator (`num`) that contains all of the digits and a denominator (`denom`) that contains some power of 10. For example, 123.456 would be represented with a numerator containing 123456 and a denominator containing 1000 (10^3).

The in-memory representation always has an unsigned denominator. We provide signed and unsigned representations that use signed and unsigned numerators, respectively. One can choose the number of bits used for both numerator and denominator in the same way as for the integer types, thus there are four types for signed values and four types for unsigned values:

```
typedef struct { Pint8   num; Puint8  denom; } Pfpoint8;
typedef struct { Pint16  num; Puint16 denom; } Pfpoint16;
typedef struct { Pint32  num; Puint32 denom; } Pfpoint32;
typedef struct { Pint64  num; Puint64 denom; } Pfpoint64;

typedef struct { Puint8  num; Puint8  denom; } Pufpoint8;
typedef struct { Puint16 num; Puint16 denom; } Pufpoint16;
typedef struct { Puint32 num; Puint32 denom; } Pufpoint32;
typedef struct { Puint64 num; Puint64 denom; } Pufpoint64;
```

There are two macros to help one use values of these in-memory types. `P_FPOINT2FLOAT32(fp)` calculates `fp.num/fp.denom` as a `Pfloat32`, while `P_FPOINT2FLOAT64(fp)` calculates `fp.num/fp.denom` as a `Pfloat64`.

4.2 Base Type Mask

The mask for base types is just an integer type, treated as an array of bits:

```
typedef Puint32 Pbase_m;
```

Masks for accessing individual bits of the base type masks are described in sections describing operations that use mask. (*cf.* Section 3.16.3). More information about how base types handle masks is available in Appendix B

4.3 Base Type Parse Descriptor

Parse descriptors for base types contain only the fields described in Section 3.13. Specific error codes are discussed when the base type read functions are described.

4.4 Character Sets

PADS currently supports two character sets for external data, ASCII and EBCDIC. As discussed in Section 3.5, the library discipline contains a field `def_charset` that selects which character set to use when one is not specified explicitly. As a result, for each 'kind' of data that has an external form made up of characters (characters, strings, character-based dates, character-based integer and floating point numbers, *etc.*),

PADS has three types: a type that indicates the external form is always ASCII, a type that indicates the external form is always EBCDIC, and a type that indicates that the external form uses the character set specified in `def_charset`.

In each section describing character-based types, we give a three-column table indicating the type(s) that use ASCII, EBCDIC, or DEFAULT character sets. For example, the next section begins with a table showing types `Pa_char` (ASCII), `Pe_char` (EBCDIC), `Pchar` (DEFAULT).

4.5 Character Base Types

4.5.1 Fixed-width character-based encoding

ASCII	EBCDIC	DEFAULT
`Pa_char`	`Pe_char`	`Pchar`

For example, writing

```
Pa_char c;
```

in a PADS source file (within a Pstruct, for example) indicates that a single ASCII character is expected. Writing a constraint such as

```
Pe_char  c : c == 'A' || c == 'B';
```

indicates that an EBCDIC capital letter A or B is expected. **NB:** Note that the constraint expression is applied to the *in-memory* representation of c, which is an ASCII value, thus the C character constants (ASCII constants) are used to specify letters A and B.

4.5.2 Special character counting base types

ASCII	EBCDIC	DEFAULT
`Pa_countX`	`Pe_countX`	`PcountX`
`Pa_countXtoY`	`Pe_countXtoY`	`PcountXtoY`

Unlike all other base types, these counting base types never advance the IO cursor. You can think of these types as "peeking ahead" to see how many occurrences of a given character appear forward of the current IO cursor position.

The PcountX types count the number of occurences of character X between the current IO cursor position and the fist EOR (end of record) or EOF (end of file). They take three parameters, `x`, `eor_required`, and `count_max`. `x` is the character to count. If `eor_required` is non-zero, then encountering EOF before EOR produces an error. If `count_max` is non-zero, EOR/EOF must be encountered before scanning `count_max` characters, otherwise an error is returned. For example,

```
Pa_countX(: '=', 0, 0 :) my_count;
```

will count the number of ASCII equals-sign characters between the IO cursor and the next EOR or EOF, with no limit on the maximum scan distance.

4.6 String Base Types (including dates and times)

The large number of string base types arises from the fact that there are many different ways to indicate the extent of a string. The entire input (up to end-of-file or end-of-record) is a sequence of bytes that can be

included in a string, so when specifying a string type in a PADS description, we need to indicate how much of that input we would like included in the string.

4.6.1 Pstring_FW

ASCII	EBCDIC	DEFAULT
Pa_string_FW	Pe_string_FW	Pstring_FW

One of the simplest ways to specify the extent of a string is to give the exact number of characters, or width, that will be included in the string. For example,

```
Pstring_FW(: 10 :) my_string;
```

Specifies a string with width 10. In this case the default character set will determine whether ASCII or EBCDIC characters are expected in the input stream. Regardless of the input character set, the resulting in-memory Pstring contains ASCII characters.

An error occurs if the specified width is not available. See Appendix B for details.

4.6.2 Pstring

ASCII	EBCDIC	DEFAULT
Pa_string	Pe_string	Pstring

For the Pstring type one specifies a 'stop character' that is expected immediately *following* the string. The extent of the string is all characters from the IO cursor up to but not including the first occurrence of the stop char. For example,

```
Pe_string(:'|':) my_string;
```

Indicates that a series of EBCDIC characters is expected, followed by an EBCDIC vertical bar. Note that the stop char is always specified in ASCII (in this case usng a C character constant). When the character set that is being read from the input is EBCDIC, the read function looks for the EBCDIC character that is equivalenet to the specified ASCII character.

4.6.3 Pstring_ME

ASCII	EBCDIC	DEFAULT
Pa_string_ME	Pe_string_ME	Pstring_ME

For type Pstring_ME, a regular expression called the *matching expression* is given, and the extent of the string is the longest sequence of characters starting at the current IO position which match this expression. Note that when you specify a regular expression as a C string, you must use two backslashes to indicate a single backslash character (otherwise C will think you are applying the special backslash operator to the following character). In a language such as Perl, which does not have this requirement, you might write

```
/\S*/
```

to create a regular expression which will match a sequence of zero or more non-space characters. To use the same regular expression in a PADSL description with the Pa_string_ME type, you would write:

```
Pa_string_ME(: "/\\S\*/" :) my_string;
```

This will match a sequence of zero or more non-space characters and assign it to `my_string`. Note that if a space occurs immediately, a match still occurs, since we specified that zero characters was OK; `my_string` would be a string of length zero. The extent is bound be end-of-record/end-of-file, so if there are no spaces before an end-of-record, `my_string` will end up containing all characters remaining in the record.

As a concrete example, if the input at the current IO cursor is `hello world` when the above string declaration is used to read from the input, `my_string` will end up containing `hello`. Note that the space that follows `hello` is *not* included, since it does not part of the match.

4.6.4 Pstring_SE

ASCII	EBCDIC	DEFAULT
`Pa_string_SE`	`Pe_string_SE`	`Pstring_SE`

For type `Pstring_SE`, a regular expression called the *stop expression* is given, and the extent of the string is the longest sequence of characters starting at the current IO position such that the characters immediately following successfully match the stop expression. None of the characters matching the stop expression are included in the result. For example,

```
Pa_string_SE(: "/\\s|$/" :) my_string;
```

The stop expression will match *either* a space character (due to the backslash-s) *or* end-of-record/end-of-file (due to the special dollar-sign character). As a result, `my_string` will end up containing all non-space characters up to (but not including) the first space characer that is found, or up to the end of the current record if no space character is found.

You may have noticed that the `Pa_string_ME` and `Pa_string_SE` examples actually specify exactly the same extent. Because of the power of regular expressions, it is often the case that you can choose to use either type. You should use whichever type results in a clearer description of what is expected in the input. (In this case, the`Pa_string_ME` form is simpler and therefore clearer.)

4.6.5 Timestamp_explicit

ASCII	EBCDIC	DEFAULT
`Pa_timestamp_explicit_FW`	`Pe_timestamp_explicit_FW`	`Ptimestamp_explicit_FW`
`Pa_timestamp_explicit`	`Pe_timestamp_explicit`	`Ptimestamp_explicit`
`Pa_timestamp_explicit_ME`	`Pe_timestamp_explicit_ME`	`Ptimestamp_explicit_ME`
`Pa_timestamp_explicit_SE`	`Pe_timestamp_explicit_SE`	`Ptimestamp_explicit_SE`

A timestamp is a combination of a calendar date and a time of day. The corresponding in-memory representation is a `Puint32` which represents the number of seconds since `00:00:00 1-Jan-1970 UTC`, also knows as "seconds since the epoch." Thus, the time `00:00:20 1-Jan-1970 UTC` would be represented internally as the number 1200, since it occurs 1200 seconds (20 minutes) past the epoch.

If the input is ``midnight Jan 1 1970'' and the time zone is UTC, then this produces a value of 0 since this is actually the epoch. If the time zone is EST, then this produces 5 * 60 * 60 since midnight in the EST timezone occurred 5 hours after the start of the epoch.

If the input explicitly has a time zone, as in ``midnight Jan 1 1970 UTC'' then the time zone in the input is used, so this would produce 0, regardless of the time zone specified for the type. Of course, not all timestamp input formats allow you to explicitly give the time zone!

The input values are ASCII or EBCDIC strings. Each `Ptimestamp_explicit` type takes as first argument the same form of specifying the string's extent as the corresponding `Pstring` type, and takes as second argument a timestamp format string which describes what the input string should contain.

Timestamp formats consists of literal characters that are are simply expected to be present in the input and special combinations of a percent-sign and a character used to indicate expected parts of the timestamp. For example, the input format `"%Y-%m-%d+%H:%M"` indicates that a format that starts with a four digit year, then a dash, then a two digit month, then a dash, then a two-digit day, then a plus sign, then a two digit hour, then a colon, then a two digit minutes. (To specify that a literal percent sign must appear in the input, use two percent signs in a row.) A full description of supported formats appears on the webpage: `www.research.att.com/~gsf/man/man3/tm.html`

Each of the `Ptimestamp_explicit` types corresponds to one of the `Pstring` types that has already been described, where each takes one additional argument to specify the input format. For example,

```
Pa_timestamp_explicit(: '|' , "%Y-%m-%d+%H:%M", P_cstr2timezone("-0500"):) my_timestamp;
```

Reads an ASCII string, up to but not including a vertical bar, and converts that string into a `Puint32` timestamp. The conversion will be successful only if the string has the specified format.

Some timestamp formats include explicit time zone information,such as the one above. PADS provides the function `P_cstr2timezone` to convert a string representation of a time zone into an value of type `Tm_zone_t *`. This function is described in Chapter 14.

For the rest,the input time zone is taken from the PADS discipline field `disc->in_time_zone`, as described in Section 15.1.10.

4.6.6 Timestamp

ASCII	EBCDIC	DEFAULT
Pa_timestamp_FW	Pe_timestamp_FW	Ptimestamp_FW
Pa_timestamp	Pe_timestamp	Ptimestamp
Pa_timestamp_ME	Pe_timestamp_ME	Ptimestamp_ME
Pa_timestamp_SE	Pe_timestamp_SE	Ptimestamp_SE

The timestamp types are the same as the `timestamp_explicit` types, except no timestamp format is given. Instead, the PADS discipline field `disc->in_formats.timestamp` is used for all `Ptimestamp` types.

4.6.7 Date_explicit

ASCII	EBCDIC	DEFAULT
Pa_date_explicit_FW	Pe_date_explicit_FW	Pdate_explicit_FW
Pa_date_explicit	Pe_date_explicit	Pdate_explicit
Pa_date_explicit_ME	Pe_date_explicit_ME	Pdate_explicit_ME
Pa_date_explicit_SE	Pe_date_explicit_SE	Pdate_explicit_SE

Dates are calendar days (no time of day). Like timestamps, we represent a date as a `Puint32` recording "seconds since the epoch."

The `Pdate_explicit` types take a second argument, a date format, which accepts the same special characters as the `Ptimestamp_explcit` types. So, technically there is nothing to stop you from using the date types to input time of day fields. However, we encourage you to use the `Ptimestamp` types when both a calendar day and a time of day are to be input, and to use the `Pdate` types when just the calendar day is to be input.

4.6.8 Date

ASCII	EBCDIC	DEFAULT
Pa_date_FW	Pe_date_FW	Pdate_FW
Pa_date	Pe_date	Pdate
Pa_date_ME	Pe_date_ME	Pdate_ME
Pa_date_SE	Pe_date_SE	Pdate_SE

The date types are the same as the `date_explicit` types, except no date format is given. Instead, the PADS discipline field `disc->in_formats.date` is used for all `Pdate` types.

4.6.9 Time_explicit

ASCII	EBCDIC	DEFAULT
Pa_time_explicit_FW	Pe_time_explicit_FW	Ptime_explicit_FW
Pa_time_explicit	Pe_time_explicit	Ptime_explicit
Pa_time_explicit_ME	Pe_time_explicit_ME	Ptime_explicit_ME
Pa_time_explicit_SE	Pe_time_explicit_SE	Ptime_explicit_SE

Times give the time of day, with no calendar date. They are represented as a `Puint32` recording seconds since midnight. For examle, the time 1am is represented as 3600 (i.e., 3600 seconds, or 60 minutes, after midnight).

The `Ptime_explicit` types take a second argument, a time format, which accepts the same special characters as the timestamp and date types. However, we encourage you to use the `Ptimes` types when just a time of day is expected.

4.6.10 Time

ASCII	EBCDIC	DEFAULT
Pa_time_FW	Pe_time_FW	Ptime_FW
Pa_time	Pe_time	Ptime
Pa_time_ME	Pe_time_ME	Ptime_ME
Pa_time_SE	Pe_time_SE	Ptime_SE

The time types are the same as the `time_explicit` types, except no time format is given. Instead, the PADS discipline field `disc->in_formats.time` is used for all `Ptime` types.

4.6.11 IP

ASCII	EBCDIC	DEFAULT
Pa_ip_FW	Pe_ip_FW	Pip_FW
Pa_ip	Pe_ip	Pip
Pa_ip_ME	Pe_ip_ME	Pip_ME
Pa_ip_SE	Pe_ip_SE	Pip_SE

The `Pip` type reads an IP address string from the input that is in numeric dotted form (as in `10.1.0.17`) using ASCII or EBCDIC digits and periods (dots). The string consists of up to four parts with values between 0 and 255, separated by periods, with an optional trailing period. When there are fewer than four parts, the missing parts are treated as implicitly zero, and are inserted as shown in the following diagram, which shows the eight legal input forms and the equivalent expanded form.

```
<part1>                                 ⟶    <part1>.0.0.0
<part1>.                                ⟶    <part1>.0.0.0.
<part1>.<part4>                         ⟶    <part1>.0.0.<part4>
<part1>.<part4>.                        ⟶    <part1>.0.0.<part4>.
<part1>.<part2>.<part4>                 ⟶    <part1>.<part2>.0.<part4>
<part1>.<part2>.<part4>.                ⟶    <part1>.<part2>.0.<part4>.
<part1>.<part2>.<part3>.<part4>         ⟶    same
<part1>.<part2>.<part3>.<part4>.        ⟶    same
```

Each <part> is made up of 1 to 3 digits which specify a number in the range [0, 255].

The result is a single `Puint32` value with each part encoded in one of the four bytes. part1 is stored in the high-order byte, part4 in the low-order byte. You can obtain each part using the macro

```
P_IP_PART(addr, part)
```

where `part` must be an integer between 1 and 4.

The digits and the "." char are read as EBCDIC chars if the EBCDIC form is used or if the default form is used and `pads->disc->def_charset` is `Pcharset_EBCDIC`. Otherwise the data is read as ASCII chars.

4.7 Integer Base Types

4.7.1 Fixed-width character-based encoding

ASCII	EBCDIC	DEFAULT
Pa_int8_FW	Pe_int8_FW	Pint8_FW
Pa_int16_FW	Pe_int16_FW	Pint16_FW
Pa_int32_FW	Pe_int32_FW	Pint32_FW
Pa_int64_FW	Pe_int64_FW	Pint64_FW
Pa_uint8_FW	Pe_uint8_FW	Puint8_FW
Pa_uint16_FW	Pe_uint16_FW	Puint16_FW
Pa_uint32_FW	Pe_uint32_FW	Puint32_FW
Pa_uint64_FW	Pe_uint64_FW	Puint64_FW

The above types are used when the input representation for an integer is a fixed number of ASCII or EBCDIC characters. The `int` types are signed types, while the `uint` types are unsigned. The number in the type name specifies how many bits are used in the in-memory represenation, thus a `Puint32` is a 32-bit (4 byte) representation of an unsigned integer.

The characters in the input can have an optional plus or minus sign for signed types, or an optional plus signed for unsigned types, followed by a set of one or more digits. In addition, leading or trailing whitespace can occur, but only if the PADS discipline field `disc->flags` has the `WSPACE_OK` flag set. The data is read as EBCDIC chars if an EBCDIC form (such as `Pe_int8`) is used, or if the default form (`Pint8`) is used and `pads->disc->def_charset` is `Pcharset_EBCDIC`. Otherwise, the data is read as ASCII chars.

4.7.2 Variable-width character-based encoding

ASCII	EBCDIC	DEFAULT
Pa_int8	Pe_int8	Pint8
Pa_int16	Pe_int16	Pint16
Pa_int32	Pe_int32	Pint32
Pa_int64	Pe_int64	Pint64
Pa_uint8	Pe_uint8	Puint8
Pa_uint16	Pe_uint16	Puint16
Pa_uint32	Pe_uint32	Puint32
Pa_uint64	Pe_uint64	Puint64

The expected input for these types is an optional sign character followed by a sequence of digits. The number of characters that make the input is variable: after the first digit, the digits are read until (but not including) the first non-digit or EOR/EOF. If the PADS discipline field `disc->flags` has the `WSPACE_OK` flag set, then leading whitespace is allowed.

4.7.3 Raw binary encoding

RAW
Pb_int8
Pb_int16
Pb_int32
Pb_int64
Pb_uint8
Pb_uint16
Pb_uint32
Pb_uint64

These are the first binary types described in this chapter. The input is not made up of ASCII or EBCDIC characters that need to be interpreted to see what number they are describing. Instead, the number itself is encoded in binary form, as a sequence of bytes.

There are binary types for signed or unsigned *binary* integers of common bit widths (8, 16, 32, and 64 bit widths). `Pb_int8` corresponds to one byte of input, `Pb_int16` to two bytes of input, and so on.

The representation in memory is just the corresponding signed or unsigned type, thus `Pb_uint16` has representation type `Puint16`. The bytes from the input are simply copied into the bytes that make up the representation. If the endian-ness of input data is different from the endian-ness of the machine, then the byte order is reversed to form the in-memory representation; otherwise the byte order is preserved.

The endian-ness of the machine running the PADS program is fixed: it is determined automatically by the PADS libary. The input data endianess is described by PADS disipline field `disc->d_endian`.

In some cases it is possible to have PADS determine the proper setting for `disc->d_endian` automatically, by using the annotation `Pendian` with the first multi-byte binary integer field that appears in the data. For example, consider this header definition:

```
Pstruct header {
   Pendian Pb_uint16 version : version < 10;
        ...
};
```

This PADS description indicates the first value in the header is a 2-byte unsigned binary integer, `version`, whose value should be less than ten. The Pendian annotation indicates that there should be two attempts at

reading the version field: once with the current `disc->d_endian` setting, and (if the read fails) once with the opposite `disc->d_endian` setting. If the second read succeeds, then the new `disc->d_endian` setting is retained, otherwise the original setting is retained.

Note that the `Pendian` pragma is only able to determine the correct endian choice for a field that has an attached constraint, where the wrong choice of endian setting will always cause the constraint to fail. (In the above example, if a value less than ten is read with the wrong `d_endian` setting, the result is a value that is much greater than ten.)

4.7.4 Serialized binary encoding

SBL	SBH
`Psbl_int8`	`Psbh_int8`
`Psbl_int16`	`Psbh_int16`
`Psbl_int32`	`Psbh_int32`
`Psbl_int64`	`Psbh_int64`
`Psbl_uint8`	`Psbh_uint8`
`Psbl_uint16`	`Psbh_uint16`
`Psbl_uint32`	`Psbh_uint32`
`Psbl_uint64`	`Psbh_uint64`

These types describe signed or unsigned binary integers that have been encoded with a specified number of bytes K. For the `PPsbl_` types, the first byte on the input stream is treated as the low-order byte of the K byte value, For the `PPsbh_` types, the first byte on the input stream is treated as the high-order byte of the K byte value, For example, `Psbl_int32(:3:)` describes a 3 byte binary encoding where the first byte encountered is the low-order byte.

These types are more general than the simpler `Pb_` types because you explicitly specify the number of bytes (from 1 to 8) independently of the target in-memory type, allowing for types such as the `Psbl_int32(:3:)` type just described. These types also explicitly specify the endian-ness of the data bytes, rather than using `disc->d_endian`.

The following table shows those cases where serialized binary types have equivalent simple binary types.

Serialized Binary Type	Equivalent type if disc->d_endian is PbigEndian	PlittleEndian
Psbl_int8(:1:)		Pb_int8
Psbl_int16(:2:)		Pb_int16
Psbl_int32(:4:)		Pb_int32
Psbl_int64(:8:)		Pb_int64
Psbl_uint8(:1:)		Pb_uint8
Psbl_uint16(:2:)		Pb_uint16
Psbl_uint32(:4:)		Pb_uint32
Psbl_uint64(:8:)		Pb_uint64
Psbh_int8(:1:)	Pb_int8	
Psbh_int16(:2:)	Pb_int16	
Psbh_int32(:4:)	Pb_int32	
Psbh_int64(:8:)	Pb_int64	
Psbh_uint8(:1:)	Pb_uint8	
Psbh_uint16(:2:)	Pb_uint16	
Psbh_uint32(:4:)	Pb_uint32	
Psbh_uint64(:8:)	Pb_uint64	

4.7.5 EBC encoding

EBC
Pebc_int8
Pebc_int16
Pebc_int32
Pebc_int64
Pebc_uint8
Pebc_uint16
Pebc_uint32
Pebc_uint64

These types describe signed or unsigned EBCDIC numeric encoded integers with a specified number of digits. **N.B.:** the specified number of digits must be odd if the value on disk can be negative. For example, Pebc_int32(:5:) describes a 5 digit signed integer.

Each byte on disk encodes one digit (using the low 4 bits). For signed values, the final byte encodes the sign (high 4 bits == 0xD for negative). E.g., a signed or unsigned 5 digit value is encoded in 5 bytes.

The legal range of values for the number of digits, num_digits, depends on target type:

Type	num_digits	Min / M
Pint8	$1-3$	P_MIN_
Puint8	$1-3$	0 / P_MA
Pint16	$1-5$	P_MIN_
Puint16	$1-5$	0 / P_MA
Pint32	$1-10$	P_MIN_
Puint32	$1-10$	0 / P_MA
Pint64	$1-19$	P_MIN_
Puint64	$1-20$	0 / P_MA

4.7.6 BCD encoding

BCD
Pbcd_int8
Pbcd_int16
Pbcd_int32
Pbcd_int64
Pbcd_uint8
Pbcd_uint16
Pbcd_uint32
Pbcd_uint64

These types describe signed or unsigned BCD numeric encoded integers with a specified number of digits. **N.B.:** the specified number of digits must be odd if the value on disk can be negative. For example, `Pbcd_int32(:5:)` describes a 5 digit signed integer.

Each byte on disk encodes two digits, 4 bits per digit. For signed values, a negative number is encoded by having number of digits be odd so that the remaining low 4 bits in the last byte are available for the sign. (low 4 bits == `0xD` for negative). A signed or unsigned 5 digit value is encoded in 3 bytes, where the unsigned value ignores the final 4 bits and the signed value uses them to get the sign.

The legal range of values for the number of digits, `num_digits`, depends on target type:

Type	num_digits	Min / M
Pint8	$1-3$	P_MIN_
Puint8	$1-3$	0 / P_MA
Pint16	$1-5$	P_MIN_
Puint16	$1-5$	0 / P_MA
Pint32	$1-11$**	P_MIN_
Puint32	$1-10$	0 / P_MA
Pint64	$1-19$	P_MIN_
Puint64	$1-20$	0 / P_MA

** **Note:** For type `Pbcd_int32` only, even though the min and max int32 have 10 digits, we allow `num_digits == 11` due to the fact that 11 is required for a 10 digit negative value. (An actual 11 digit number would cause a range error, so the leading digit must be 0.)

4.8 Floating Point Base Types

4.8.1 Variable-width character-based encoding

ASCII	EBCDIC	DEFAULT
Pa_float32	Pe_float32	Pfloat32
Pa_float64	Pe_float64	Pfloat64

These types describe ASCII or EBCDIC character-based encodings of floating point numbers. The input representation must have this form:

```
[+|-]DIGITS[.][DIGITS][(e|E)[+|-]DIGITS]
```

Where `DIGITS` is a sequence of one or more digit characters, `(e|E)` indicates either a lower- or upper-case letter 'E', and elements in square brackets are optional. Note that there must be at least one digit before the (optional) dot (period) character.

If the input has a valid sequence of input characters that make up a float, then the float is converted to a `Pfloat32` or `Pfloat64`, according to the type. For example, if you specify a `Pa_float32` then a characters making up a float will be read from the input and converted to an in-memory `Pfloat32`.

4.9 Fixed Point Base Types

The following types encode a numerator value on the input stream in different formats, as described below. They all produce an in-memory `Pfpoint` value whose denominator is determined from the second type argument, `d_exp`, where the denominator is implicitly 10^{d_exp} and is not encoded on disk.

The legal range of values for `d_exp` depends on the target in-memory type:

Type	d_exp	
`Pfpoint8 / ufpoint8`	$0-2$	
`Pfpoint16 / ufpoint16`	$0-4$	
`Pfpoint32 / ufpoint32`	$0-9$	
`Pfpoint64 / ufpoint64`	$0-19$	`10,0`

4.9.1 Serialized binary encoding

SBL	SBH
`Psbl_fpoint8(K,` d_{exp}`)`	`Psbh_fpoint8(K,` d_{exp}`)`
`Psbl_fpoint16(K,` d_{exp}`)`	`Psbh_fpoint16(K,` d_{exp}`)`
`Psbl_fpoint32(K,` d_{exp}`)`	`Psbh_fpoint32(K,` d_{exp}`)`
`Psbl_fpoint64(K,` d_{exp}`)`	`Psbh_fpoint64(K,` d_{exp}`)`
`Psbl_ufpoint8(K,` d_{exp}`)`	`Psbh_ufpoint8(K,` d_{exp}`)`
`Psbl_ufpoint16(K,` d_{exp}`)`	`Psbh_ufpoint16(K,` d_{exp}`)`
`Psbl_ufpoint32(K,` d_{exp}`)`	`Psbh_ufpoint32(K,` d_{exp}`)`
`Psbl_ufpoint64(K,` d_{exp}`)`	`Psbh_ufpoint64(K,` d_{exp}`)`

These types describe fixed-point numbers where the numerator is encoded in serialized binary form on the input stream. Serialized binary encodings are described above for the `Psbl_` and `Psbh_` integer types. Like those integer types, the number of bytes on the input is specified as the first type argument. The legal range of values for the number of bytes depends on target type, and follows the same rule specified for the `Psbl_` and `Psbh_` integer types.

Each type takes a second argument, `d_exp`, where the denominator value is implicitly 10^{d_exp} and is not encoded on disk. For example, `sbl_fpoint32(:3, 2:)` specifies that a numererator is encoded on the input as a three binary bytes with the low-order byte appearing first, where the resulting in-memory `Pfpoint32` has the value read from the input as its numerator, and the number 10^2 (i.e., 100) as its denominator.

4.9.2 EBC encoding

EBC
Pebc_fpoint8
Pebc_fpoint16
Pebc_fpoint32
Pebc_fpoint64
Pebc_ufpoint8
Pebc_ufpoint16
Pebc_ufpoint32
Pebc_ufpoint64

These types describe fixed-point numbers where the numerator is encoded as EBCDIC numeric digits on the input stream. This encoding is described above for the `Pebc_` integer types. Like those integer types, the number of digits on the input is specified as the first type argument. The legal range of values for the number of digits depends on target type, and follows the same rule specified for the `Pebc_` integer types.

Each type takes a second argument, `d_exp`, where the denominator value is implicitly 10^{d_exp} and is not encoded on disk. For example, `ebc_fpoint32(:3, 2:)` specifies that a numererator is encoded on the input as three EBCDIC digits, where the resulting in-memory `Pfpoint32` has the value read from the input as its numerator, and the number 10^2 (i.e., 100) as its denominator.

4.9.3 BCD encoding

BCD
Pbcd_fpoint8
Pbcd_fpoint16
Pbcd_fpoint32
Pbcd_fpoint64
Pbcd_ufpoint8
Pbcd_ufpoint16
Pbcd_ufpoint32
Pbcd_ufpoint64

These types describe fixed-point numbers where the numerator is encoded as BCD numeric digits on the input stream. This encoding is described above for the `bcd_` integer types. Like those integer types, the number of digits on the input is specified as the first type argument. The legal range of values for the number of digits depends on target type, and follows the same rule specified for the `bcd_` integer types.

Each type takes a second argument, `d_exp`, where the denominator value is implicitly 10^{d_exp} and is not encoded on disk. For example, `bcd_fpoint32(:3, 2:)` specifies that a numererator is encoded on the input as three BCD digits, where the resulting in-memory `Pfpoint32` has the value read from the input as its numerator, and the number 10^2 (i.e., 100) as its denominator.

Chapter 5

Pstructs

Pstructs are used to describe sequences of values with potentially unrelated types. Intuitively, they correspond to record-like structures externally and C-`structs` in memory.

5.1 Syntax

The syntax for **Pstruct**s is given by the following BNF grammar fragment:

qualifier ::= **Pomit** | **Pendian**
qualifiers ::= *qualifier* | *qualifier qualifiers*
constraint ::= : *predicate*
ty ::= *c_ty* | *p_ty*
full_field ::= [*qualifiers*] *p_ty* identifier [*constraint*] ; [*p_comment*]
comp_field ::= **Pcompute** [**Pomit**] *ty* identifier = *expression* [*constraint*] ;
literal_field ::= *p_coreliteral*;
array_field ::= [*qualifiers*] *p_ty* '[' *p_size_spec* ']' identifier [: *p_array_constraints*] ; [*p_comment*]
opt_field ::= [*qualifiers*] **Popt** *p_ty* identifier [: *opt_predicates*] ; [*p_comment*]
field ::= *full_field* | *comp_field* | *literal_field* | *array_field* | *opt_field*
fields ::= *field* | *field fields*
struct_ty ::= **Pstruct** identifier [*p_formals*] {
fields
} [**Pwhere** { *predicate* }] ;

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment are defined elsewhere. Predicates (*predicate*) are described in Section 3.3. PADS types (*p_ty*) and formal parameters (*p_formals*) are described in Section 3.6. PADS comments (*p_comment*) are described in Section 3.2. Core literals (*p_coreliteral*) are described in Section 3.4. For in-line arrays, size specifications (*p_size_spec*) and array constraints appear in Chapter 7. Option constraints (*opt_predicates*) are defined in Section 9.1. Expressions (*expression*) represent any C expression, while *c_ty* denotes any C type.

5.1.1 Example

The following **Pstruct** describes the request portion of a common-log format web-server log, an example of which is:

```
GET /research.att.com/projects/PADS/index.html HTTP/1.0
```

```
Pstruct http_request_t {
  '\"'; http_method_t   meth;     /- Method used during request
  ' ';  Pstring(:' ':)  req_uri;  /- Requested uri.
  ' ';  http_v_t        version : checkVersion(version, meth);
                                  /- HTTP version
  '\"';
};
```

The **Pstruct** http_request_t has full fields meth, req_uri, and version that use the (omitted) auxiliary types http_method_t, Pstring, and http_v_t to describe the HTTP method, URI, and version formats, respectively. It has literal fields '\"' and ' ' to describe the quotations and spaces in the external representation. The version field uses the C function checkVersion:

```
int  checkVersion(http_v_t version, http_method_t meth) {
  if ((version.major == 1) && (version.minor == 0)) return 1;
  if ((meth == LINK)  || (meth == UNLINK )) return 0;
  return 1;
}
```

to ensure that the obsolete HTTP methods LINK and UNLINK are only used with HTTP version 1.0.

5.1.2 Full fields

Each full field in a **Pstruct** must include the name of the field and its type. The name serves to document the data and to permit later reference. The type determines how that piece of the **Pstruct** will be processed. Optionally, each full field may be preceeded by a qualifier sequence *cf.* Section 5.1.6.

Each full field may be followed by a constraint (*cf.* Section 3.3). Such a constraint is used to express the conditions under which a properly parsed value of the field type is a legal value for the field. The field itself and all earlier fields in the **Pstruct** are in scope in the constraint, as are any parameters to the **Pstruct**. In the example, the checkVersion predicate on the version field uses the values of the meth and version fields to determine if the version value is legal. If the constraint associated with a field evaluates to false (*i.e.*, zero) after parsing, then the parse descriptor returned with the in-memory representation will indicate a user-constraint violation has occurred for the field.

Each full field in a **Pstruct** may optionally be followed by a PADS comment. Such comments are reflected by the PADS compiler into the output library as comments.

5.1.3 Computed fields

Instead of being read from an external source, the value of a computed field is set from an initializing expression. Such fields are marked by the **Pcompute** keyword. Each such field gives its name and the type to be included in the in-memory representation. If the given type is a PADS type, the field will behave exactly as if it were read from the external source. With a C type, some services may not be available in the generated library, such as automatic accumulation and printing. Each computed field also gives a C expression

to initialize the field. This expression must have the type declared for the field. Previously read fields in the **Pstruct** and any parameters to the **Pstruct** are in scope in this expression. Like full fields, computed fields admit the **Pomit** qualifier and may have an associated constraint.

The `computeExample` **Pstruct** sets the value of its computed field `index` from the full field `base` and the `offset` parameter.

```
Pstruct computeExample(:int offset:){
  Pint32 base;
  Pcompute int index = base + offset;
};
```

5.1.4 Literal fields

Literal fields can be character, string, or regular expression literals. They are written using the notation described in Section 3.4.

In addition to specifying literals to consume from the external representation, literal fields also play a role in error recovery. If the generated parser encounters a syntactic error while parsing a full field, causing it to enter panic mode (*cf.* Section 3.10), the parser will scan to find the next literal, marking all intervening fields as errors in the associated parse descriptor. The library discipline has parameters that allow the library user to tune the extent of such scanning (*cf.* Section 15.1.5).

5.1.5 In-line declarations

For conciseness, PADS allows anonymous option and array types to be declared within **Pstruct** field declarations.

Array declarations

In-line array declarations include a size specification after the type of the field. For example, the following **Pstruct** matches a resolved IP address (of the form `135.27.24.12`) and an integer recording a number of bytes, separted by a vertical bar:

```
 Pstruct log_t {
  Puint8 [4] ip  : Psep('.') && Pterm(Pnosep); /- resolved ip address
  '|';
  Puint32 numBytes;
};
```

After the field name, PADS permits an optional colon followed by array constraints. Details about size specifications, array constraints, and the in-memory representation of arrays may be found in Chapter 7.

Option declarations

In-line options are marked by the keyword **Popt**. For example, the following **Pstruct** matches two optional integers separated by a vertical bar and terminated by a newline.

```
Precord Pstruct entry2{
  Popt Puint32 f;
  '|';
  Popt Puint32 g;
}
```

This declaration is equivalent to the `entry1` type defined in Section 9.1.1. Fields with in-line option declarations admit the option form of constraints, which are described in Section 9.1.2. As an example, the **Pstruct** `entry4`

```
Precord Pstruct entry4{
  Popt Puint32 x1 : Psome i => { i % 2 == 0};
  Popt Puint32 x2 : Psome i => { i % 2 != 0};
  '|';
  Popt Puint32 y1 : Psome i => { i % 2 == 0};
  Popt Puint32 y  : Psome i => { i % 2 != 0};
  '|';
};
```

uses option constraints to specify when the option should match. Type `entry4` is equivalent to the type `entry3` defined in Section 9.1.1 .

Details about the in-memory representation of options appear in Section 9.2.2.

5.1.6 Qualifiers

Non-literal fields can take one or more qualifiers.

Pomit This qualifier indicates that the field should not be included in the in-memory representation of the **Pstruct**. Because they are not included in-memory, omitted fields cannot be accumulated or printed.

Pendian During initialization, the PADS library determines the endian-ness of the underlying machine and stores the result in the library handle. Each library handle discipline stores the endian-ness of the data being parsed, initially assuming the endian-ness of the data matches that of the machine. The **Pendian** qualifier directs the generated parser to check the endian-ness of the data; it can only be used in the presence of a user constraint. The qualifier causes the parser to read the field and check the associated constraint. If the constraint is violated, the bytes associated with the field are swapped, and the constraint is re-tested. If this second attempt succeeds, the endian-ness of the data is toggled in the library discipline. The value of the data endian-ness flag can also be set programmatically (*cf.*Section 15.1.8).

5.1.7 Optional **Pwhere** clause

If given, a **Pwhere** clause expresses constraints over the entirety of a **Pstruct** value. The values of all previous fields and any parameters to the **Pstruct** are in scope. Within the context of a **Pparsecheck** clause, constants `begin` and `end`, each of type `Ppos_t` are available. Constant `begin` is bound to the input position of the beginning of the **Pstruct**; `end` is bound to its end. If the predicate given in the **Pwhere** clause evaluates to false (*i.e.*, zero), the error code in the associated parse descriptor will indicate a user-constraint error has occurred.

The **Pwhere** clause in the `whereExample` **Pstruct** ensures that the sum of the first two fields is less than the given limit.

```
Pstruct whereExample(:int limit:){
  Pint32 first;
  Pint32 second;
} Pwhere {first + second < limit;};
```

5.2 Generated library

5.2.1 In-memory representation

The in-memory representation of a **Pstruct** is a C struct of the same name. Each field of the C struct corresponds to a full or computed field of the **Pstruct**. The type of each full field in the C struct is the in-memory representation of the PADS type associated with the field. The type of each computed field is the given C type.

The C type `http_request_t` is the in-memory representation of the PADS type of the same name.

The type `Pstring` is the in-memory representation of the base type `Pstring` (*cf.* Chapter 4). Note that literal fields do not appear in the in-memory representation.

5.2.2 Mask

The mask of a **Pstruct** with name `myStruct` is a C struct with name `myStruct_m`. For each full field in `myStruct`, there is a corresponding field in the mask struct, the type of which is the mask type for the field. In addition, there is a `structLevel` field, which has the base mask type. This field allows library users to toggle operations at the level of the structure as a whole.

For example, the mask type `http_request_t_m` has the following structure:

```
typedef struct http_request_t_m_s http_request_t_m;

struct http_request_t_m_s {
  Pbase_m structLevel;
  http_method_t_m meth;         /* nested constraints */
  Pbase_m req_uri;              /* nested constraints */
  http_v_t_m version;           /* nested constraints */
  Pbase_m version_con;          /* struct constraints */
};
```

5.2.3 Parse descriptor

The parse descriptor of a **Pstruct** with name `myStruct` is a C struct with name `myStruct_pd`. This struct has the fields described in Section 3.13. In addition, for each full field in `myStruct`, there is a corresponding field in the parse descriptor struct, the type of which is the parse descriptor type for the field.

For example, the parse descriptor type `http_request_t_pd` has the following structure:

```
Perror_t http_request_t_init (P_t *pads,http_request_t *rep);

Perror_t http_request_t_pd_init (P_t *pads,http_request_t_pd *pd);

Perror_t http_request_t_cleanup (P_t *pads,http_request_t *rep);

Perror_t http_request_t_pd_cleanup (P_t *pads,http_request_t_pd *pd);

Perror_t http_request_t_copy (P_t *pads,http_request_t *rep_dst,
                              http_request_t *rep_src);

Perror_t http_request_t_pd_copy (P_t *pads,http_request_t_pd *pd_dst,
                                 http_request_t_pd *pd_src);

void http_request_t_m_init (P_t *pads,http_request_t_m *mask,
                            Pbase_m baseMask);

Perror_t http_request_t_read (P_t *pads,http_request_t_m *m,
                              http_request_t_pd *pd,http_request_t *rep);

ssize_t http_request_t_write2buf (P_t *pads,Pbyte *buf,size_t buf_len,int *buf_full,
                                  http_request_t_pd *pd,http_request_t *rep);

ssize_t http_request_t_write2io (P_t *pads,Sfio_t *io,
                                 http_request_t_pd *pd,http_request_t *rep);

int http_request_t_verify (http_request_t *rep);

int http_request_t_genPD (P_t *pads, http_request_t *rep,
                          http_request_t_pd *pd);
```

Figure 5.1: Prototypes of operations generated for the **Pstruct** httpRequest.

```
typedef struct http_request_t_pd_s http_request_t_pd;

struct http_request_t_pd_s {
  Pflags_t pstate;
  Puint32 nerr;
  PerrCode_t errCode;
  Ploc_t loc;
  http_method_t_pd meth;
  Pbase_pd req_uri;
  http_v_t_pd version;
};
```

5.2.4 Operations

The operations generated by the PADS compiler for a **Pstruct** are those described in Chapter 3. For the **Pstruct** http_request_t, the prototypes of the generated functions appear inFigure 5.1

Read function

The error codes for **Pstruct**s are:

Code	Meaning
P_NO_ERR	Indicates no error occurred
P_STRUCT_FIELD_ERR	Indicates that an error occurred during parsing one of the fields of the **Pstruct**. The parse descriptor for each full field with an error will contain more information describing the precise nature of the error.
P_STRUCT_EXTRA_BEFORE_SEP	Indicates that there were unexpected data before a literal field in the **Pstruct**.
P_MISSING_LITERAL	Indicates that the read function failed to find a literal field

If multiple errors occur during the parsing of a **Pstruct**, the errCode field will reflect the first detected error. The parse descriptors for nested pieces will describe any errors detected while reading those pieces.

Warning: At the moment, read functions do not check that all referenced data in constraint expressions are meaningful before checking the constraint. Referenced data might be meaningless either because there was an error parsing earlier data or because the supplied mask directed the read function to skip the field.

Accumulator functions

Accumulator functions for **Pstruct**s are described in Chapter 16.

Histogram functions

Histogram functions for **Pstruct**s are described in Chapter 17.

Clustering functions

Clustering functions for **Pstruct**s are described in Chapter 18.

Chapter 6

Punions

Punions are used to express variations in data. PADS supports two forms of union: switched and in-place. The first form supports data sources where there is an indication (*i.e.*, a switch) in the data prior to the union indicating which alternative should be chosen. The second form supports data sources where no such switch is present. The default for this case is to try the branches in order until one parses without any errors. There is also a qualifier (**Plongest**) that indicates the parser should take the branch that consumes the most input.

6.1 Syntax

union_field ::= *full_field* | *comp_field* | *literal_field* | *array_field* | *opt_field* | **Pempty**

branch ::= **Pcase** *expression* : *union_field* | **Pdefault**: *union_field*

branches ::= *branch* | *branch branches*

switched ::= **Pswitch** (*expression*){ *branches* }

in_place ::= *union_field* | *union_field in_place*

union_bdy ::= *switched* | *in_place*

union_ty ::= [**Plongest**] **Punion** identifier [*p_formals*] {
union_bdy
} [**Pwhere** { *predicate* }] ;

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment were defined previously, as follows. Full fields (*full_field*), in-line array declarations (*array_field*), and in-line option declarations (*option_field*) appear in Section 5.1.2, computed fields (*comp_field*) in Section 5.1.3, literals (*literal_fields*) in Section 3.4 and PADS parameter lists (*p_formals*) in Section 3.6. Expressions (*expression*) represent any C expression.

6.1.1 Example: Switched Punions

The PADS declarations in Figure 6.1 describe data which uses an integer tag to determine the format of the rest of the data. The **Pstruct** `choice` specifies that the integer field `which` should be passed to the switched **Punion** `branches`. The `branches` declaration describes three alternatives, depending upon the value of the tag `which`. A tag value of `1` indicates an unsigned integer will follow:

```
Punion branches(:Puint32 which:) {
  Pswitch (which) {
    Pcase 1   : Pint32                number : number % 2 == 0;
    Pcase 2   : Pstring_SE(:"EOR":)   name;
    Pdefault : Pcompute Puint32       other = which;
  }
}

Precord Pstruct choice{
  Puint32           which;
  branches(:which:) branch;
}
```

Figure 6.1: Switched **Punion** for describing data variations determined by tags earlier in the data source.

```
1 4
```

while a tag value of 2 indicates a string terminated by an end-of-record mark:

```
2 hello
```

Any other value for the tag will fall into the default clause of the union, which indicates that no further data exists:

```
3
```

6.1.2 Example: Switched Punions with Pempty

The PADS declarations in Figure 6.2 are similar to those in Figure 6.1 except that instead of computing a default value for the default case, this declaration uses the special type **Pempty**, which indicates that there is nothing on disk for this case.

6.1.3 Example: In-place Punions

The following in-place **Punion** describes a data fragment that is either a resolved or a symbolic IP address:

```
Punion clihost_t  {
  nIP resolved;    /- 135.207.23.32
  sIP symbolic;    /- www.research.att.com
};
```

The (omitted) types nIP and sIP describe named and symbolic IP addresses, respectively. The comments embedded in the description give an example of each of the two forms. With in-place **Punion**s, the parser tries each of the branches in turn until it finds one that matches the data without any errors.

6.1.4 Example: In-place Punions with Pempty

The following in-place **Punion** describes a data fragment that is either an integer or nothing using the special type **Pempty**.

```
Punion body_t(:Puint8 j:) {
  Pswitch (j) {
    Pcase 0 : Puint32 i;
    Pcase 1 : Pchar   c;
    Pdefault : Pempty;
  }
};

Precord Pstruct entry_t {
  Puint8 i;
  ':';
  body_t(:i:) body;
};
```

Figure 6.2: Switched **Punion** with **Pempty** default.

```
Punion Choice_t {
  Puint32 i;
  Pempty;
};
```

6.1.5 Switched Punions

Pswitch

The expression on which a switched **Punion** branches can be any C expression of integer type (as in a **switch** statement in C). Typically, this expression is computed from a parameter to the switched **Punion**.

Branches

The body of a switched **Punion** is a non-empty sequence of branches. Each branch in a switched **Punion** can have one of two forms: a **Pcase** statement, or a **Pdefault** statement. The **Pcase** form specifies an integer value. During parsing, the first **Pcase** expression whose value equals the **Pswitch** expression is selected as the active description. If no **Pcase** expression matches, the **Pdefault** expression (if present) matches instead.

6.1.6 Union fields

The body of each branch in a switched **Punion** is a *union_field*, while the body of each in-place **Punion** is a non-empty sequence of such fields. There are six varieties of union fields, five borrowed from **Pstruct**s: full fields (Section 5.1.2), computed fields (Section 5.1.3), in-place array declarations (Section 5.1.5), in-place option declarations (Section 5.1.5), and literal fields, and **Pempty**, which is special to **Punion**s. The only semantic change from **Pstruct**s is that in **Punion**s, earlier fields are not in scope for later fields because only one branch of a union can be active at a time. A **Pempty** branch corresponds to nothing in the physical representation.

The name of a branch in either kind of union is the name of the declared identifier for full and computed fields. For literal fields, it is the literal itself, unless the programmer specifies a different name using the **Pfrom** form. The **Pfrom** form must be used when the literal is not a valid C identifier. For example, the following in-place union uses the **Pfrom** form to provide names for the string literal `"*"` and the regular expression literal **Pre** `"/a+/"`.

```
Precord Punion test {
  "baz";
  'c';
  star Pfrom("*");
  as Pfrom(Pre "/a+/");
  Pint32 f;
};
```

Within a **Punion**, the type **Pvoid** describes no physical data. There is no corresponding field in the in-memory representation of the union, but the union tag is set to indicate which branch of the union was used during the parse. This type is useful with switched unions where the value of the tag indicates that there is no data associated with the tag. For example, the following **Punion** `branches`

```
Punion branches(:Puint32 which:) {
  Pswitch (which) {
    Pcase 0   : Pvoid                noValue;
    Pcase 1   : Pint32               someValue;
  }
}
```

indicates that when the value of the tag `which` is `0`, there is no more data associated with the type `branches` in the input source, while the tag value `1` indicates there is a `Pint32` following.

6.1.7 In-line declarations

In-line declarations in **Punion**s have the same form as in **Pstruct**s *cf.* Section 5.1.5.

6.1.8 `Plongest`

By default, in-place **Punion**s commit to the first branch that parses without any errors. Adding the **Plongest** qualifier to the **Punion** declaration, however, indicates that the parser should instead select the *longest* match, *i.e.*, the match that consumes the most input.

6.1.9 Optional `Pwhere` clause

If given, a **Pwhere** clause expresses constraints over the entirety of a **Punion** value. Special constants `tag` and `val` are in scope, of the tag and value types for the union, respectively (*cf.* Section 6.2.1). The first indicates which branch of the union matched, while the second contains the representation of the matched branch. Within the context of a **Pparsecheck** clause, constants `begin` and `end`, each of type `Ppos_t` are available. Constant `begin` is bound to the input position of the beginning of the **Punion**; `end` is bound to its end. If the predicate given in the **Pwhere** clause evaluates to false (*i.e.*, zero), the error code in the associated parse descriptor will indicate a user-constraint error has occurred.

6.2 Generated library

6.2.1 Tags

In addition to the types generated for every PADS specification, the PADS compiler generates an extra type declaration for every **Punion**: a enumerated type with one component for each branch in the union, plus

```
typedef union branches_u_u branches_u;

union branches_u_u {
  Pint32 number;                /* number % 2 == 0 */
  Pstring name;
  Puint32 other;                /* other = which */
};

typedef struct branches_s branches;

struct branches_s {
  branches_tag tag;
  branches_u val;
};
```

Figure 6.3: Type declarations related to the in-memory representation of the **Punion** branches.

an extra component corresponding to a match failure. The names of the tags correspond to the names of the branches in the union, unless that name has already been defined elsewhere. In this case, the name of the tag is unionName_branchName. The name of the tag enumeration is the name of the PADS specification with the _tag suffix. For example, the generated enumeration type for the **Punion** branches is the following:

```
typedef enum branches_tag_e branches_tag;

enum branches_tag_e {
  branches_err=0,
  number=1,
  name=2,
  other=3
  };
```

6.2.2 In-memory representation

The in-memory representation of both forms of **Punion** is a C struct containing a tag field to indicate which branch of the union has been populated followed by a val field storing the union itself. We represent unions as C unions, with one component per non-literal branch of the union. The representation-related type declarations for the **Punion** branches appear in Figure 6.3.

6.2.3 Mask

The mask of a **Punion** is a C struct. For each full and computed field in the union, there is a corresponding field in the mask, the type of which is the type of the mask type for that field. For example, the mask type branches_m has the following structure:

```
typedef struct branches_m_s branches_m;

struct branches_m_s {
  Pbase_m unionLevel;
  Pbase_m number;               /* nested constraints */
  Pbase_m number_con;           /* union constraints */
  Pbase_m name;                 /* nested constraints */
};
```

```
typedef union branches_pd_u_u branches_pd_u;

union branches_pd_u_u {
  Pbase_pd number;
  Pbase_pd name;
  Pbase_pd other;               /* other = which */
};

typedef struct branches_pd_s branches_pd;

struct branches_pd_s {
  Pflags_t pstate;
  Puint32 nerr;
  PerrCode_t errCode;
  Ploc_t loc;
  branches_tag tag;
  branches_pd_u val;
};
```

Figure 6.4: Type declarations related to the parse descriptor for the **Punion** branches.

Union masks have one additional field unionLevel that allows the programmer to toggle behavior at the level of the union as a whole.

6.2.4 Parse descriptor

The parse descriptor of a **Punion** is a C struct, with all the fields described in Section 3.13. In addition, there is a tag field indicating which branch of the union was populated during parsing and a val field which stores the parse descriptor of the populated branch, represented as a C union. The parse descriptor declarations corresponding to the PADS type branches appear in Figure 6.4.

6.2.5 Operations

The operations generated by the PADS compiler for a **Punion** are those described in Chapter 3. In addition, there is an extra function that converts a value of the tagtype for the union to a string. For a **Punion** named myUnion, this function has the name myUnion_tag2str. For the **Punion** branches, the prototypes for all the generated functions appear in Figure 6.5.

Read function

The read function for a switched **Punion** evaluates the **Pswitch** expression and uses a C switch statement to jump to the appropriate branch. If there is no **Pdefault** branch and none of the **Pcase** branches match, the read function will return the error code P_UNION_MATCH_ERR and set the tag fields of the parse descriptor and in-memory representation to the error tag.

For an in-place **Punion** without the **Plongest** qualifier, the read function speculatively reads each branch in turn until it finds one that parses without errors. Before reading a branch, the read function marks the current location in the input. It then tries to read the data described by the type of the branch. If the nested read function succeeds and any user-level constraint on the branch also succeeds, the read function commits to the parse, sets the tag fields to the name of the successful branch, and returns P_NO_ERR. If an error occurs, the read function aborts the read, rolling back the input to the marked location, and tries

```
char const *branches_tag2str (branches_tag which);

Perror_t branches_init (P_t *pads,branches *rep);

Perror_t branches_pd_init (P_t *pads,branches_pd *pd);

Perror_t branches_cleanup (P_t *pads,branches *rep);

Perror_t branches_pd_cleanup (P_t *pads,branches_pd *pd);

Perror_t branches_copy (P_t *pads,branches *rep_dst,branches *rep_src);

Perror_t branches_pd_copy (P_t *pads,branches_pd *pd_dst,
                           branches_pd *pd_src);

void branches_m_init (P_t *pads,branches_m *mask,Pbase_m baseMask);

Perror_t branches_read (P_t *pads,branches_m *m,Puint32 which,
                        branches_pd *pd,branches *rep);

ssize_t branches_write2buf (P_t *pads,Pbyte *buf,size_t buf_len,int *buf_full,
                            Puint32 which,branches_pd *pd,branches *rep);

ssize_t branches_write2io (P_t *pads,Sfio_t *io,Puint32 which,
                           branches_pd *pd,branches *rep);

int branches_verify (branches *rep,Puint32 which);

int branches_genPD (P_t *pads,branches *rep, branches_pd *pd, Puint32 which);
```

Figure 6.5: Prototypes of operations generated for the **Punion** branches.

the next branch. If the last branch in an in-place **Punion** fails, the read function returns the error code P_UNION_MATCH_ERR and sets the tag fields of the parse descriptor and in-memory representation to the error tag. Errors that occur duing parsing branches of an in-line **Punion** are surpressed because of the speculative nature of the parsing.

For in-place **Punion**s with the **Plongest** qualifier, the read function parses each branch, preserving in the in-memory representation the non-erroneous branch that occupied the most space in the physical source. Ties are resolved in favor of the earliest branch. Non-erroneious branches that occupy zero space in the physical representation, such as **Pcompute** fields or non-matching options, are considered matches; the first such branch will be preserved in the in-memory representation if no longer non-erroneous branch is found. The read function reports an error if no branch matches without an error.

The error codes for **Punion**s are:

Code	Meaning
P_NO_ERR	Indicates no error occurred
P_UNION_MATCH_ERR	Indicates that no branch of the union parsed without error.

Accumulator functions

Accumulator functions for **`Punion`**s are described in Chapter 16.

Histogram functions

Histogram functions for **`Punion`**s are described in Chapter 17.

Clustering functions

Clustering functions for **`Punion`**s are described in Chapter 18.

Chapter 7

Parrays

Parrays are used to describe sequences of values of the same type. We call the repeated type the *base type* of the sequence.

7.1 Syntax

The syntax for **Parray**s is given by the following BNF grammar fragment:

p_size_spec ::= [*expresssion*] | [*expression*] : [*expression*]

p_term_expression ::= **Pnosep** | *p_expression*

p_array_constraint ::= **Psep**(*p_expression*) | **Pterm**(*p_term_expression*)
| **Plast**(*predicate*) | **Pended**(*predicate*)
| **Plongest** | **Pomit**(*predicate*)

p_array_constraints ::= *p_array_constraint* | *p_array_constraint* && *p_array_constraints*

p_range ::= '[' *expression* .. *expression*']' | identifier
p_forall ::= **Pforall** (identifier **Pin** *p_range* : *expression*)
p_array_post ::= *predicate* | *p_forall*
p_array_posts ::= *p_array_post* | *p_array_post* && *p_array_posts*

array_ty ::= **Parray** identifier [*p_formals*] {
p_ty '['*p_size_spec*']' [: *p_array_constraints*]
} [**Pwhere** { *p_array_posts* }] ;

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment were defined previously. Predicates (*predicate*) are described in Section 3.3. PADS types (*p_ty*) and formal parameters (*p_formals*) are described in Section 3.6. PADS expressions *p_expressions* are defined in Section 3.15. Literals (*p_literal*) are described in Section 3.4. Expressions (*expression*) represent any C expression. We put single quotation marks around the left and right brackets to indicate they appear in the grammer, rather than as a meta-notation for optionality.

7.1.1 Examples

In this section, we illustrate various uses of **Parray**s.

Resolved IP address

To describe a resolved IP address in ASCII, an example of which is:

```
135.207.26.22
```

we use the specification:

```
Parray nIP {
  Puint8 [4] : Psep('.') && Pterm(' ');
};
```

which indicates that **Parray** nIP is a sequence of four Puint8's. The elements of the sequence are separated by dots and the sequence is terminated by a space. As a part of parsing the sequence, the generated read function for this type will read the separators. It will check that the terminator is present, but will not consume it. This specification has two termination conditions: a maximum size (4) and a terminator (a space). Parsing will terminate when either condition is satisified. An error will be reported if the other condition is not also satisifed.

Binary sequence of integers

As another example, the **Parray** seq_t

```
Parray seq_t{
  Pb_int32 [] : Plast(elts[current] > 10);
};
```

uses the **Plast** predicate to terminate a sequence of 32-bit binary integers as soon as one of those integers is greater than ten. The special variable elts refers to the sequence matched so far, while current is the index of the most recently read element. If the expression within the **Plast** clause evaluates to true, parsing for the array terminates. The current element is then the last element in the array.

Sorted sequence of structs

In the next example, the **Parray** sorted_t uses a **Pwhere** clause to check that the elements of the sequence were sorted by the id field of the element type..

```
Precord Pstruct elem_t{
  Puint32 id;
  '|';
  Puint32 val;
};

Parray sorted_t(: Pint32 size :) {
  elem_t [size];
} Pwhere {
  Pforall( i Pin [0..length-2] : elts[i].id < elts[i+1].id );
};
```

A **Pforall** expression executes its body once for each value in the range. The index variable, `i` in the example, is bound to the range value in the body.

7.1.2 Special variables

Within the various expression contexts of **Parray** declarations, a number of variables are in scope. The following table lists the variables, their types, and their bindings.

Variable	Type	Contexts	Binding
`numRead`	`int`	**Pparsecheck**	Number of elements read from source.
`length`	`int`	all	Number of elements in in-memory representation of sequence.
`elts`	`rep_t *`	all	In-memory representation of element sequence.
`pds`	`pd_t *`	all	In-memory representation of parse descriptor sequence.
`current`	`int`	all	Index of most recently read element.
`elt`	`rep_t`	all	Most recently read element of sequence.
`pd`	`pd_t`	**Pparsecheck**	Most recently set parse descriptor.
`consume`	`int`	**Pended**	See **Pended** section for explanation.
`begin`	`Ppos_t`	**Pparsecheck**	Position in input source before reading sequence.
`end`	`Ppos_t`	**Pparsecheck** within **Pwhere**	Position in input source after reading sequence.
`eltBegin`	`Ppos_t`	**Pparsecheck**	Position in input source before reading element.
`eltEnd`	`Ppos_t`	**Pparsecheck**	Position in input source after reading element.

In the table, we use `rep_t` to denote the type of the in-memory representation of the base type of the sequence *i.e.*, `rep_t` is the base type for the sequence. Similarly, `pd_t` denotes the type of the parse descriptor for the base type of the sequence. In the table, *arrayName* denotes the name of the array; it is a synonym for the special variable `elts`.

7.1.3 Size specifications

There are five different kinds of size specifications possible:

`[]`	unbounded
`[n]`	exactly size `n`
`[low:high]`	at least `low`, at most `high`
`[low:]`	at least `low`, no upper bound
`[:high]`	at most `high`, no lower bound

As an example, the following specification describes an integer sequence of length at least `min` and at most `max`:

```
Parray list(:Puint32 min, Puint32 max:) {
  Pint32 [min : max] : Psep('|');
};
```

A maximum size specification constitutes a termination condition for reading the array. It is an error for the lower bound to be greater than the upper bound.

7.1.4 Psep

The **Psep** constraint is used to specify separators, *i.e.* data in the source that occurs between sequence elements. The body of the **Psep** constraint can be a character, string, or regular expression. It is an error for a **Parray** to contain more than one **Psep** clause.

7.1.5 Pterm

The **Pterm** constraint is used to specify a terminator, *i.e.* data in the source that occurs after the last element of the sequence and indicates that the sequence has ended. The body of the **Pterm** constraint can be a character, string, or regular expression, or the special keyword **Pnosep**, which indicates that the lack of a separator should be interpreted as signalling the end of the sequence. The terminator is not consumed by the read function for the array. It is an error for a **Parray** to contain more than one **Pterm** clause. The following description uses **Pnosep** to indicate that an IP address should be terminated by the lack of a period.

```
Parray nIP2 {
  Puint8 [4] : Psep('.') && Pterm(Pnosep);
};
```

7.1.6 Plast

The **Plast** constraint is used to specify an arbitrary termination condition. The body of the constraint is a predicate expression (*cf.* 3.3). It is evaluated after reading each element in the array. If it evaluates to true, array parsing terminates. Section 7.1.2 lists the variables that are in scope within the predicate. The following description uses a **Plast** predicate to terminate array parsing if the number of input bytes consumed by reading the sequence is greater than a specified amount.

```
Pstruct character_string {
  Psbh_uint8(:1:)        length;
  Pa_string_FW(:length:) bytes;
};

Parray TXT_t(:Puint16 rdlength:) {
  character_string [] :
      Plast(Pparsecheck(eltEnd.offset - begin.offset >= rdlength));
};
```

If an array has a **Plast** constraint, it cannot have a **Plongest** or a **Pended** constraint.

7.1.7 Plongest

The **Plongest** specifier indicates that parsing the sequence continues until the parser detects an error, either in parsing a separator for the array or in parsing an element. The parser returns the data that produced the error to the input. If the array declaration includes a separator, the separator preceding the returned element is also returned to the input. The following PADS code describes a non-empty sequence of even integers separated by the character ('a'), followed by a character ('a'), followed by a non-empty sequence of odd integers, separated by the character ('a') and terminated by the character ('b'):

```
int isDone(Pint32 value, Pbase_pd p, int *consume){
  if (p.errCode != P_NO_ERR) return 0;  /* continue sequence */
  if (value == 1) {
    *consume = 1;       /* consume value */
    return 1;           /* terminate sequence */
  };
  if (value == -1){
    *consume = 0;       /* return value to input */
    return 1;           /* terminate sequence */
  };
  return 0;             /* continue sequence */
};

Parray fseq_t {
  Pint32 [] : Psep(',')
              && Pended(Pparsecheck(isDone(fseq_t[current],
                                           pds[current], &consume)));
};
```

Figure 7.1: **Parray** with **Pomit** clause.

```
Ptypedef Pint32 even_t : even_t x => {x % 2 == 0};
Ptypedef Pint32 odd_t  : odd_t  x => {x % 2 == 1};

Parray eseq_t {
  even_t [1:] : Psep('a') && Plongest;
};

Parray oseq_t {
  odd_t [1:] :  Psep('a') && Plongest && Pterm('b');
};

Precord Pstruct entry{
  eseq_t evens;
  'a';
  oseq_t odds;
  'b';
};
```

If an array has a **Plongest** constraint, it cannot have a **Plast** or a **Pended** constraint.

7.1.8 Pended

The **Pended** clause is similar to **Plast**, except that it allows the specification writer to indicate whether to consume the terminating element. As with **Plast**, **Pended** takes a predicate and has in scope the variables describd in Section 7.1.2. When the predicate returns true, the array terminates. By default, the terminating element is returned to the input, as is any preceeding separator. To indicate that the terminating element should instead be considered part of the array, the predicate can set the special variable `consume` to true as a side-effect.

The specification in Figure 7.1 describes a sequence of comma-separated integers. The `isDone` function in the **Pended** clause examines the most recently read value and the parse descriptor to determine if the sequence has terminated. Because the function takes the parse descriptor as an argument, it must be within the scope of a **Pparsecheck** clause.

If an array has a **Pended** constraint, it cannot have a **Plast** or a **Plongest** constraint.

7.1.9 Pomit

Like **Plast** and **Pended**, the **Pomit** clause takes a predicate and has the variables in scope described in Section 7.1.2. The predicate is evaluated after reading each element in the array. When the predicate returns true, the just-read element is not stored into the array; it is "omitted". It is because of the **Pomit** predicate that the special variables `numRead` and `length` need not be the same. Variable `numRead` indicates the total number of possible elements that have been read, while `length` indicates the number that have been stored into the array. Consequently, `numRead` will always be greater than or equal to `length`.

The following code reads a sequence of up to four space-separated, signed integers terminated by the end of the record. It omits each of the negative numbers from the in-memory representation.

```
Parray nseq_t{
  Pint32 [:4] : Psep(' ') && Pomit(elt < 0) && Pterm(Peor);
};
```

7.1.10 Optional Pwhere clause

If given, a **Pwhere** clause expresses constraints over the entirety of a **Parray** value. The special variables that are in scope are described in Section 7.1.2. If the predicate given in the **Pwhere** clause evaluates to false (*i.e.*, zero), the error code in the associated parse descriptor will indicate a user-constraint error has occurred. Syntactically, **Pwhere** clauses for arrays are a `&&`-separated sequence of array predicates. There are two types of array predicates: general predicates, which include **Pparsecheck** constraints, and **Pforall** predicates. The value of the **Pwhere** clause is the conjunction of the values of each of the array predicates. The predicates are evaluated in the order they are listed.

7.1.11 General predicates

General predicates, which include **Pparsecheck** predicates, are described in Section 3.3

7.1.12 Pforall

Pforall predicates provide a way to write constraints over the entirety of the in-memory representation of an array after it has been fully parsed. The following example checks that the resulting integer sequence is in sorted order

```
Precord Parray intList {
  Puint32 [INTLIST_SIZE] : Psep('|');
} Pwhere {
    Pforall( i Pin [0..length-2] : (intList[i] < intList[i+1]) )  &&
    Pparsecheck(end.offset - begin.offset > 10);
};
```

Syntactically, the **Pforall** clause gives an index variable (`i` in the example) and a range for that variable (`0` to `length-2`, inclusive). Ranges can be written either as an integer lower and upper bound, as in the example, or as the name of the array, in which case the lower bound is zero and the upper bound is one less than the length of the array. Following the colon, it gives a boolean-valued expression. This expression

is executed once for each value of the index in the range; the value of the clause is the conjunction of the resulting values.

7.1.13 In-line arrays

For conciseness, **Parray**s can be declared in-line in **Pstruct** and **Punion** declartions (*cf.* Section 5.1.5 and Section 6.1.7).

7.2 Termination conditions

There are a number of conditions under which parsing for an array will terminate:

- *Size limit.* If an array specification includes an upper bound, then parsing will terminate after the indicated number of elements have been placed in the in-memory representation. Note that omitted elements are not included in this total.
- *Terminator.* For an array with a **Pterm** clause, parsing terminates whenever the terminating sequence is found.
- *End-of-source.* Parsing terminates if it reaches the end of the data source.
- **Plast**. If an array has a **Plast** clause, parsing terminates whenever the specified predicate evalutes to true.
- **Plongest**. For an array with a **Plongest** clause, parsing terminates as soon as there is an error detected in reading an element. The erroneous element value is not included in the array.
- **Pended**. If an array has a **Pended** clause, parsing terminates whenever the specified predicate evaluates to true.
- *Failure to consume input.* If array parsing code fails to consume any input in reading the next element of the sequence, then parsing for the array will terminate. (The alternative is to loop infinitely).

A single array specification can include many of these termination conditions. If any of the conditions trigger, then array parsing terminates.

After a termination condition has been detected, the parse function does error checking. In particular, if an array specification has a minimum size and the length of the in-memory representation does not satisfy this requirement, the parse descriptor will record the error `P_ARRAY_SIZE_ERR`.

7.3 Generated library

To explain the data structures and operations generated for array declarations, we will use the description of resolved IP addresses as an example.

```
Parray nIP {
  Puint8 [4] : Psep('.') && Pterm(' ');
};
```

7.3.1 In-memory representation

The in-memory representation of a **Parray** is a struct with three fields.

```
typedef struct nIP_s nIP;

struct nIP_s {
  Puint32 length;
  Puint8 *elts;
  RBuf_t *_internal;
};
```

The first of these is an integer recording the length of the array in memory. The second is an array of elements of the base type of the array. The third field is a pointer to a growable buffer that manages the dynamically-allocated space for the array.

7.3.2 Mask

The mask for a **Parray** is a struct containing a pair of masks. The `element` field stores a mask of the base element mask type. This mask specifies how to handle each element in the array. The `arrayLevel` mask allows the user to toggle behavior at the level of the **Parray** as a whole.

```
typedef struct nIP_m_s nIP_m;

struct nIP_m_s {
  Pbase_m element;              /* per-element */
  Pbase_m arrayLevel;           /* entire array */
};
```

7.3.3 Parse descriptor

The parse descriptor for a **Parray** is a struct, with all the fields described in Section 3.13.

```
typedef struct nIP_pd_s nIP_pd;

struct nIP_pd_s {
  Pflags_t pstate;
  Puint32 nerr;                 /* Number of array errors */
  PerrCode_t errCode;
  Ploc_t loc;
  Puint32 neerr;                /* Number of element errors */
  Puint32 firstError;           /* if errCode == ARRAY_ELEM_ERR,
                                   index of first error */
  Puint32 numRead;              /* Number of elements read */
  Puint32 length;               /* Number of elements in memory */
  Pbase_pd *elts;
  RBuf_t *_internal;
};
```

In addition, the field `neerr` records the number of errors that occurred while processing elements of the array. The field `firstError` records the index of the first element that triggered an error. This field is

```
Perror_t nIP_init (P_t *pads,nIP *rep);

Perror_t nIP_pd_init (P_t *pads,nIP_pd *pd);

Perror_t nIP_cleanup (P_t *pads,nIP *rep);

Perror_t nIP_pd_cleanup (P_t *pads,nIP_pd *pd);

Perror_t nIP_copy (P_t *pads,nIP *rep_dst,nIP *rep_src);

Perror_t nIP_pd_copy (P_t *pads,nIP_pd *pd_dst,nIP_pd *pd_src);

void nIP_m_init (P_t *pads,nIP_m *mask,Pbase_m baseMask);

Perror_t nIP_read (P_t *pads,nIP_m *m,nIP_pd *pd,nIP *rep);

int nIP_verify (nIP *rep);

int nIP_genPD (P_t *pads, nIP *rep, nIP_pd *pd);
```

Figure 7.2: Prototypes of operations generated for the **Parray** nIP.

only valid if the errCode has the value ARRAY_ELEM_ERR. The numRead field records the number of elements in the array storing the parse descriptors for each of the array elements. The field elts stores the array of base parse descriptors. Finally, field _internal contains a pointer to the growable buffer that manages the dynamically-allocated space for the array of base parse descriptors.

7.3.4 Operations

The operations generated by the PADS compiler for a **Parray** are those described in Chapter 3. Figure 7.2 lists these operations.

Read function

The error codes for **Pstruct**s appear in Figure 7.3. If multiple errors occur during the parsing of a **Parray**, the errCode field will reflect the first detected error. The array of parse descriptors will describe any errors detected while reading the **Parray** elements.

Accumulator functions

Accumulator functions for **Parray**s are described in Chapter 16.

Histogram functions

Histogram functions for **Parray**s are described in Chapter 17.

Clustering functions

Clustering functions for **Parray**s are described in Chapter 18.

Code	Meaning
P_NO_ERR	No error occurred
P_ARRAY_ELEM_ERR	An error occurred during parsing one of the elements of the **Parray**. The parse descriptor for the element whose index is `firstError` will contain more information describing the precise nature of the error.
P_ARRAY_SEP_ERR	Error reading a separator.
P_ARRAY_TERM_ERR	Error reading the terminator.
P_STRUCT_EXTRA_BEFORE_SEP	Unexpected data before a separator.
P_STRUCT_EXTRA_BEFORE_TERM	Unexpected data before the terminator.
P_ARRAY_SEP_TERM_SAME_ERR	The separator and terminator sequences were the same.
P_ARRAY_SIZE_ERR	The size of the parsed **Parray** did not match the specified size constraints.
P_ARRAY_MIN_BIGGER_THAN_MAX_ERR	Minimum size larger than maximum size.
P_ARRAY_MIN_NEGATIVE	Negative minimum size.
P_ARRAY_MAX_NEGATIVE	Negative maximum size.
P_ARRAY_USER_CONSTRAINT_ERR	**Pwhere** clause returned false.

Figure 7.3: Error codes that can be returned by **Parray** read functions.

Chapter 8

Penums

Penums allow a fixed collection of source constants to be converted into integers in-memory.

8.1 Syntax

p_enum_prefix	::=	**Pprefix** (identifier)
p_enum_base_ty	::=	**Pfrom** (*p_ty*)
p_enum_modifier	::=	*p_enum_prefix* | *p_enum_base_ty*
p_raw_enum_field	::=	*p_literal* [= expression]
p_enum_field	::=	*p_raw_enum_field*, [p_comment]
p_last_enum_field	::=	*p_raw_enum_field* [p_comment]
p_enum_fields	::=	*p_last_enum_field*
	|	*p_enum_field p_enum_fields*
enum_ty	::=	**Penum** identifier [*p_formals*] [*p_enum_modifier*] { *p_enum_fields* } ;

8.1.1 Example

If the physical representation consists of the following strings:`S_init`, `S_lec`, `S_care`, `S_for`, `S_if`, and `S_tpv`, then we can use the specification:

```
Penum orderStates Pprefix("S_"){
    init,
    lec = 2,
    care,
    my_for Pfrom("for") = 10,
    my_if  Pfrom("if"),
    tpv = 5
};
```

to describe the data. Because this **Penum** does not explicitly list a PADS type for the physical representation of the constants, the system assumes the underlying representation is a string. In this case, the labels in the enumeration correspond to the strings in the physical representation. To support physical string representations that are not valid C identifiers, the **Pfrom** clause allows the user to specify the physical string separately from the enumeration label, *e.g.*, `my_for` in place of `for`. The argument to the **Pfrom** clause can be any

character, string, or regular expression. The optional **Pprefix** clause indicates that all the labels in the enumeration are prefixed by the specified string in the physical representation. PADS assigns integer values to the labels in the enumeration in the same fashion that C does, so the first label is given the value zero, and each successive label is given the next higher value. The user can specify a value for a given label using the equal expression syntax. The compiler does not check that all labels are given distinct values. An artifact of the embedding into C is that all labels must be globally distinct. If two different enumerations contain the same string, one of them must be differentiated using a **Pfrom** clause.

To support the case where the physical representation of the constant is not a string, PADS allows users to specify in the *p_enum_base_ty* clause any PADS type with an integer in-memory representation as the base type of the enumeration. For example, the following code

```
Penum Bool_t Pfrom (Pb_uint8) { False, True };
```

defines a type `Bool_t` that is represented on disk as a one-byte, unsigned binary number. A physical value of `0` corresponds to the enumeration label `False`, while a value of `1` corresponds to `True`.

8.2 Generated library

We use the `orderStates` example to illustrate the data structures and functions generated for **Penum**s.

8.2.1 In-memory representation

The following C code is the generated in memory representation for the `orderStates` **Penum**. The associated values are computed at PADS compile time and so the associated values must be compile-time constants, as in C.

```
typedef enum orderStates_e orderStates;
enum orderStates_e {
  S_init=0,
  S_lec=2,
  S_care=3,
  S_my_for=10,
  S_my_if=11,
  S_tpv=5
  };
```

8.2.2 Mask

The mask for a **Penum** is simply a base mask:

```
typedef Pbase_pd orderStates_pd;
```

8.2.3 Parse descriptor

The parse descriptor for a **Penum** is simply a base parse descriptor:

```
typedef Pbase_m orderStates_m;
```

```
char const *orderStates2str (orderStates which);

Perror_t orderStates_init (P_t *pads,orderStates *rep);

Perror_t orderStates_pd_init (P_t *pads,orderStates_pd *pd);

Perror_t orderStates_cleanup (P_t *pads,orderStates *rep);

Perror_t orderStates_pd_cleanup (P_t *pads,orderStates_pd *pd);

Perror_t orderStates_copy (P_t *pads,orderStates *rep_dst,
                           orderStates *rep_src);

Perror_t orderStates_pd_copy (P_t *pads,orderStates_pd *pd_dst,
                              orderStates_pd *pd_src);

void orderStates_m_init (P_t *pads,orderStates_m *mask,
                         Pbase_m baseMask);

Perror_t orderStates_read (P_t *pads,orderStates_m *m,
                           orderStates_pd *pd,orderStates *rep);

int orderStates_verify (orderStates *rep);

int orderStates_genPD (P_t *pads, orderStates *rep, orderStates_pd *pd);
```

Figure 8.1: Prototypes of operations generated for the **Penum** `orderStates`.

8.2.4 Operations

The operations for **Penum**s are those described in Chapter 3, with one addition. The `orderStates2str` function converts the integer representation of a **Penum** into a C string. These functions appear in Figure 8.1.

Read Function

The error codes for **Penum**s are:

Code	Meaning
`P_NO_ERR`	Indicates no error occurred
`P_ENUM_MATCH_ERR`	Indicates that the no branch of the enum matched.

Accumulator functions

Accumulator functions for **Penum**s are described in Chapter 16.

Histogram functions

Histogram functions for **Penum**s are described in Chapter 17.

Clustering functions

Clustering functions for **`Penum`**s are described in Chapter 18.

Chapter 9

Popts

Popt are used to describe optional data. The in-memory representation of an option records if the data was available and if so, the value of that data.

9.1 Syntax

p_opt_some	::=	**Psome** identifier => { *predicate* }
p_opt_none	::=	**Pnone** => { *predicate* }
opt_predicates	::=	*p_opt_some* '|' *p_opt_none*
	|	*p_opt_none* '|' *p_opt_some*
	|	*p_opt_none*
	|	*p_opt_some*
opt_ty	::=	**Popt** *p_ty* identifier [*p_formals*] [: *opt_predicates*] ;

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment were defined previously, as follows. Predicates (*predicate*) are defined in Section 3.3, PADS types (*p_ty*) and parameter lists (*p_formals*) in Section 3.6.

9.1.1 Examples

The following declaration indicates that the type `oPuint32` describes an optional `Puint32`.

```
Popt Puint32 oPuint32;

Precord Pstruct entry1{
  oPuint32 f;
  '|';
  oPuint32 g;
}
```

The **Pstruct** `entry1` uses the type `oPuint32` to describe new-line terminated records with the form:

```
12|24
```

```
|23
|
24|
```

In this case, the representations of both `f` and `g` will indicate that the data matched, storing the values `12` and `24`, respectively. For the second line, the representation of `f` will record no match, while `g`'s will indicate a match with value `23`.

A slightly more complex example of options uses predicates to determine if the option matches the input data:

```
Popt Puint32 even_t : Psome i => {i % 2 == 0};
Popt Puint32 odd_t  : Psome i => {i % 2 != 0};

Precord Pstruct entry3{
  even_t x1;
  odd_t  x2;
  '|';
  even_t y1;
  odd_t  y2;
  '|';
};
```

Here, the type `even_t` matches only even `Puint32`s, while `odd_t` matches only odd `Puint32`s. The **`Psome`** clause binds the identifier `i` to the in-memory representation of the `Puint32` found in the data source (if one is found without error). With `i` bound, it executes the associated predicate, which ensures that the number is even for `even_t` and odd for `odd_t`. The type `entry3` uses these types to describe newline-terminated data of the form:

```
12|14|
13|15|
|13|
13|12|
```

For the first record, `x1` will match `12` and `y1` will match `14`, while `x2` and `y2` will be marked as not matching. For the second record, `x2` and `y2` will match instead, with `x1` and `y1` being marked as no match, *etc.*

9.1.2 Constraints

Option constraints can have a **`Psome`** clause and/or a **`Pnone`** clause. The **`Psome`** clause specifies a variable and a predicate to execute if the base type of the option is successfully read. If a legal base element is found, the variable is bound to the in-memory representation of the base element and the associated predicate is executed in that context. If the predicate returns true, the

9.1.3 In-line options

For conciseness, **`Popt`**s can be declared in-line in **`Pstruct`** and **`Punion`** declarations (*cf.* Section 5.1.5 and Section 6.1.7).

9.2 Generated library

Currently, **Popt**s are implemented by translation into **Punion** declarations with two branches. The first branch corresponds to the case where the value is present in the source. The name of this branch is `some_OptTy`, where `OptTy` is the name of the **Popt** type. The second branch corresponds to the case where the value is not present; its name is `none_OptTy`. For example, the PADS compiler translates the **Popt** declaration

```
Popt Ty OptTy;
```

into

```
Punion OptTyTrans {
  Ty                some_OptTy;
  Pcompute Puint32  none_OptTy = 0;
};
```

and then generates the appropriate code for this declaration.

9.2.1 Tags

Because the compilation of **Popt**s is based on that of **Punion**s, the PADS compiler generates a tag type to describe whether the option was present.

```
typedef enum OptTy_tag_e OptTy_tag;

enum OptTy_tag_e {
  OptTy_err=0,
  some_OptTy=1,
  none_OptTy=2
  };
```

Details about the form of this declaration may be found in Section 6.2.1.

9.2.2 In-memory representation

The in-memory representation of an option is a C struct containing a `tag` field and a `val` field. The tag indicates whether the option was present. If the tag indicates it was, the value field stores the corresponding value.

```
typedef union OptTy_u_u OptTy_u;
union OptTy_u_u {
  Ty some_OptTy;                /* value is present */
};

typedef struct OptTy_s OptTy;
struct OptTy_s {
  OptTy_tag tag;
  OptTy_u val;
};
```

9.2.3 Mask

The mask of a **Popt** is a C struct with two fields. The first field, called `unionLevel`, allows the programmer to toggle behavior at the level of the option as a whole. The second field controls the behavior of various-library functions on the "some" branch of the option, *i.e.*, the branch that corresponds to the value being present in the source.

```
typedef struct OptTy_m_s OptTy_m;

struct OptTy_m_s {
  Pbase_m unionLevel;
  Ty_m some_OptTy;              /* nested constraints */
};
```

9.2.4 Parse descriptor

The parse descriptor of a **Popt** is a C struct, with all the fields described in Section 3.13. In addition, there is a `tag` field indicating whether the option appeared in the source and a `val` field which stores the parse descriptor of the populated branch, represented as a C union.

```
typedef union OptTy_pd_u_u OptTy_pd_u;
union OptTy_pd_u_u {
  Ty_pd some_OptTy;
  Pbase_pd none_OptTy;          /* value was not present. none_OptTy = 0 */
};

typedef struct OptTy_pd_s OptTy_pd;
struct OptTy_pd_s {
  Pflags_t pstate;
  Puint32 nerr;
  PerrCode_t errCode;
  Ploc_t loc;
  OptTy_tag tag;
  OptTy_pd_u val;
};
```

9.2.5 Operations

The operations generated by the PADS compiler for a **Popt** are those described in Chapter 3. In addition, there is an extra function that converts a value of the tagtype for the option to a string. For a **Popt** named `OptTy`, this function has the name `OptTy_tag2str`. For the **Punion** `OptTy`, the prototypes for all the generated functions appear in Figure 9.2.5.

Read function

The error codes for **Popt**s are:

Code	Meaning
`P_NO_ERR`	Indicates no error occurred
`P_OPTION_MATCH_ERR`	Indicates that no branch of the option parsed without error.

```
char const *OptTy_tag2str (OptTy_tag which);

Perror_t OptTy_init (P_t *pads,OptTy *rep);

Perror_t OptTy_pd_init (P_t *pads,OptTy_pd *pd);

Perror_t OptTy_cleanup (P_t *pads,OptTy *rep);

Perror_t OptTy_pd_cleanup (P_t *pads,OptTy_pd *pd);

Perror_t OptTy_copy (P_t *pads,OptTy *rep_dst,OptTy *rep_src);

Perror_t OptTy_pd_copy (P_t *pads,OptTy_pd *pd_dst,OptTy_pd *pd_src);

void OptTy_m_init (P_t *pads,OptTy_m *mask,Pbase_m baseMask);

Perror_t OptTy_read (P_t *pads,OptTy_m *m,OptTy_pd *pd,OptTy *rep);

ssize_t OptTy_write2buf (P_t *pads,Pbyte *buf,size_t buf_len,
                         int *buf_full,OptTy_pd *pd,OptTy *rep);

ssize_t OptTy_write2io (P_t *pads,Sfio_t *io,OptTy_pd *pd,OptTy *rep);

int OptTy_verify (OptTy *rep);

int OptTy_genPD (P_t *pads, OptTy *rep, OptTy_pd *pd);
```

Figure 9.1: Prototypes of operations generated for the **Popt** TyOpt.

For a option parsing function to return the P_OPTION_MATCH_ERR, the **Popt** declaration must include a **Pnone** clause that returns false.

Accumulator functions

Accumulator functions for **Popt**s are described in Chapter 16.

Histogram functions

Histogram functions for **Popt**s are described in Chapter 17.

Clustering functions

Clustering functions for **Popt**s are described in Chapter 18.

Chapter 10

Ptypedefs

Ptypedefs allow additional constraints to be added to an existing type.

10.1 Syntax

typedef_predicates ::= identifier identifier => { *predicate* }
typedef_ty ::= **Ptypedef** *p_ty* identifier [*p_formals*] [**:** *typedef_predicates*] ;

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment were defined previously, as follows. Predicates (*predicate*) are defined in Section 3.3, PADS types (*p_ty*) and parameter lists (*p_formals*) in Section 3.6.

10.1.1 Examples

Ptypedef predicates (*typedef_predicates*) are written over a variable bound to the in-memory representation of the base type of the **Ptypedef**. For example, the declaration

```
Ptypedef Puint32 bid_t : bid_t x => {x > 100};
```

defines a new type `bid_t` to be a `Puint32` with the additional constraint that any legal bid must be greater than 100. We read this constraint "If `x` is a `bid_t`, then `x` must be greater than 100." In the body of the constraint, the variable `x` has type `bid_t`, the in-memory represention type of the **Ptypedef**.

Like all other PADS types, **Ptypedef**s can be parameterized. The declaration

```
Ptypedef Pa_uint64_FW(:len:)
   pn_t(:Puint8 len, Puint64 hi:): pn_t x => {x <= hi};
```

introduces a new type `pn_t`. The base type for this declaration is a 64-bit unsigned integer, represented in the source as a sequence of `len` ASCII digits. The declaration adds the constraint that any `pn_t` must have a value less than `hi`. In general, the constraint can be any integer-valued expression.

10.2 Generated library

We will use the `bid_t` example to illustrate the data structures and functions generated for **Ptypedef**s.

10.2.1 In-memory representation

The in-memory representation of a **Ptypedef** is the same as the representation of the underlying base type:

```
typedef Puint32 bid_t;
```

10.2.2 Mask

The mask for a **Ptypedef** is a C struct containing a pair of masks. The `base` field stores a mask of the base mask type. This mask specifies how to handle the underlying type. The `user` mask allows the user to toggle behavior at the level of the **Ptypedef** as a whole. For example, during parsing, the `base` mask lets the user control constraint checking of the base type, while the `user` mask lets the user control checking the constraint associated with the **Ptypedef** itself.

```
typedef struct bid_t_m_s bid_t_m;
struct bid_t_m_s {
  Pbase_m base;          /* Base mask */
  Pbase_m user;          /* Typedef mask */
};
```

10.2.3 Parse descriptor

The parse descriptor for a **Ptypedef** is the same as the parse descriptor for the underlying base type.

```
typedef Pbase_pd bid_t_pd;
```

10.2.4 Operations

The operations generated by the PADS compiler for a **Ptypedef** are those described in Chapter 3. The operations appear in Figure 10.1

Read function

The error codes for **Ptypedef**s are:

Code	Meaning
`P_NO_ERR`	Indicates no error occurred
`P_TYPEDEF_CONSTRAINT_ERR`	Indicates that the typedef constraint failed.

Accumulator functions

Accumulator functions for **Ptypedef**s are described in Chapter 16.

```
Perror_t bid_t_init (P_t *pads,bid_t *rep);

Perror_t bid_t_pd_init (P_t *pads,bid_t_pd *pd);

Perror_t bid_t_cleanup (P_t *pads,bid_t *rep);

Perror_t bid_t_pd_cleanup (P_t *pads,bid_t_pd *pd);

Perror_t bid_t_copy (P_t *pads,bid_t *rep_dst,bid_t *rep_src);

Perror_t bid_t_pd_copy (P_t *pads,bid_t_pd *pd_dst,bid_t_pd *pd_src);

void bid_t_m_init (P_t *pads,bid_t_m *mask,Pbase_m baseMask);

Perror_t bid_t_read (P_t *pads,bid_t_m *m,bid_t_pd *pd,bid_t *rep);

int bid_t_verify (bid_t *rep);

int bid_t_genPD (P_t *pads, bid_t *rep, bid_t_pd *pd);
```

Figure 10.1: Prototypes of operations generated for the **Ptypedef** bit_t.

Histogram functions

Histogram functions for **Ptypedef**s are described in Chapter 17.

Clustering functions

Clustering functions for **Ptypedef**s are described in Chapter 18.

Chapter 11

Ptrans

Ptrans (short for transformations) allow users to specify both a physical and a logical representation for data, both as PADS types. Users supply a pair of functions that map between the physical and logical representations. The physical type is used to parse the data, which is then transformed according to the supplied transformation function. The logical type is used in tools, such as accumulators, XML convertion, *etc.* The `write2buf` and `write2io` functions first call the reverse translations to map the logical representation to the physical representation before printing the data. This mechanism has been designed to give users some control over the logical representation of their data and to make it easier to add new base types.

11.1 Syntax

p_trans_spec ::= *expression* [*p_actuals*]
p_trans_maskmap ::= **Pmaskmap** *expression* **;**
trans_ty ::= **Ptrans** identifier [*p_formals*] {
p_trans_spec **:** *p_ty* <=> *p_ty* **:** *p_trans_spec* **;**
[*p_trans_maskmap*] }

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment were defined previously, as follows. Expressions (*expression*) are any C-expression, PADS types (*p_ty*), parameter lists (*p_formals*), and lists of actual arguments (*p_actuals*) in Section 3.6.

11.1.1 Examples

The following **Ptrans** declaration defines a type of 32-bit integers represented on disk as ASCII strings of length `size`.

```
Ptrans hexint32(:Puint32 size:){
 toInt32: Pstring_FW(:size:) <=> Puint32: toHexString(:pads, size :);
};
```

In this case, the type `Pstring_FW(:8:)` denotes the physical type while `Puint32` denotes the logical type. The function `toInt32` converts the parsed string and the corresponding parse descriptor into a 32-bit integer and parse descriptor, while the function `toHexString` performs the reverse translation. The notation `toHexString(:pads,size:)` allows the user to pass additional arguments to the translation

```
void toInt32(Pstring *src, Pbase_pd *src_pd, Puint32 *dest, Pbase_pd *dest_pd){
  size_t i;
  Puint32 digit;
  *dest = 0;
  *dest_pd = *src_pd;  /* set destination parse descriptor from source */
  for(i=0; i<src->len; i++){
    digit = hexCharToInt(src->str[i]);
    if (digit <0) {
      dest_pd->nerr++;
      dest_pd->errCode = P_TRANSFORM_FAILED;
      break;
    } else {
      *dest = *dest * 16 + digit;
      // should check for overflow here
    }
  }
};
```

Figure 11.1: Code to convert a hex string of length `size` into a `Puint32`

function. The parameter `pads` is a special variable in scope within the body of the **`Ptrans`** expression. It is bound to the PADS handle active when operations related to the **`Ptrans`** are invoked. Figures 11.1and 11.2 show the implementations of the two conversion functions. Note that the function `toHexString` needs the PADS handle to pass as an argument to the `Pstring_cstr_copy` function that copies the C string `str` into the desired `Pstring`.

11.1.2 Supplying a mask conversion function

If the masks of the logical and physical types differ, or if the desired conversion from the logical to the physical mask is not the identity function, the **`Ptrans`** declaration must specify a function to convert the logical mask to the physical mask. For example, the **`Ptrans`** `hexint32withMask` in Figure 11.3 uses the function `cnvMask` to map the mask for the logical type to one for the physical type. In the example, the mapping is simply the identify function, but the user is free to supply an arbitrary mapping function as long as it has the necessary type signature. The generated library calls this function to convert the logical mask to the physical mask just before invoking the parser for the physical representation.

11.1.3 Special variables

Within the various expression contexts of **`Ptrans`** declarations, the special variable `pads` is in scope.

Variable	Type	Contexts	Binding
`pads`	`P_t`	all	Active PADS handle.

11.2 Generated library

11.2.1 In-memory representation

The in-memory representation of a **`Ptrans`** is the same as the representation of logical type.

```
void toHexString(P_t *p, Puint32 size, Puint32 *src, Pbase_pd *src_pd,
                 Pstring *dest, Pbase_pd *dest_pd){
  Puint32 local = *src;
  int i;
  char str[256];
  Puint32 msize = (size < 256) ? size : 255;
  *dest_pd = *src_pd;

  if (msize < size) {
      // Error: hex representation too large
      dest_pd->nerr++;
      dest_pd->errCode = P_TRANSFORM_FAILED;
  }
  for (i=0; i<msize + 1; i++){
    str[i] = 0;
  }
  for (i = msize-1; i>=0; i--){
    char result = intToHexChar(local % 16);
    if ('z' == result) {
      // Error: hex string contained a non-hex digit
      dest_pd->nerr++;
      dest_pd->errCode = P_TRANSFORM_FAILED;
      break;
    }
    str[i] = result;
    local = local / 16;
  }
  if (local != 0) {
    // Error: number too big to be represented in given size.
      dest_pd->nerr++;
      dest_pd->errCode = P_TRANSFORM_FAILED;
  };
  if (P_OK != Pstring_cstr_copy(p, dest, str, size)) {
    // Error: copy into Pstring failed
      dest_pd->nerr++;
      dest_pd->errCode = P_TRANSFORM_FAILED;
  };
}
```

Figure 11.2: Code to convert a `Puint32` into a hex string of length `size`.

```
void cnvMask(Pbase_m *phy, Pbase_m *log){
  *phy = *log;
}

Ptrans hexint32withMask(:Puint32 size:){
 toInt32: Pstring_FW(:size:) <=> Puint32: toHexString(:pads, size :);
 Pmaskmap cnvMask;
};
```

Figure 11.3: **`Ptrans`** declaration with a supplied mask translation function.

11.2.2 Mask

The mask of a **`Ptrans`** is the same as the mask of the logical type.

11.2.3 Parse descriptor

The parse descriptor for a **`Ptrans`** is the same as the parse descriptor for the logical type.

11.2.4 Operations

The operations generated by the PADS compiler for a **`Ptrans`** are those described in Chapter 3.

Read function

The error codes for **`Ptrans`** are:

Code	Meaning
`P_NO_ERR`	Indicates no error occurred
`P_TRANSFORM_FAILED`	Indicates a failure during transformation.

Accumulator functions

Accumulator functions for **`Ptrans`** are just as the accumulator functions for the logical type.

Histogram functions

Histogram functions for **`Ptrans`** are just as the histogram functions for the logical type.

Clustering functions

Clustering functions for **`Ptrans`** are just as the clustering functions for the logical type.

Chapter 12

Ptry

Ptrys provide the ability to do arbitrary look-ahead. It checkpoints the data source, then parses the data at the current position using the underlying type of the **Ptry**. It sets the in-memory representation and parse descriptor for the **Ptry** as indicated by the parse, and then rolls back the data source to the check point.

12.1 Syntax

try_ty ::= **Ptry** identifier [*p_formals*] [*p_actuals*] ;

12.1.1 Example

The following code fragment defines the **Ptry** type `ForwardInt`.

```
Ptry ForwardInt Puint8_FW(:1:);

Punion VarInt(:Puint8 i:){
  Pswitch(i>5) {
    Pcase 0: Puint32 smallInt;
    Pcase 1: Puint64 largeInt;
  }
};

Pstruct entry_t{
  ForwardInt           firstDigit;
  VarInt(:firstDigit:) fullNumber;
};
```

The type `entry_t` uses `ForwardInt` to parse the first digit of an integer. The **Punion** `VarInt` switches on whether this first digit is greater than `5` to determine if the integer should be parsed as a 32- or 64-bit integer.

12.2 Generated library

12.2.1 In-memory representation

The in-memory representation of a **`Ptry`** is the same as the representation of underlying type.

12.2.2 Mask

The mask of a **`Ptry`** is the same as the mask of the underlying type.

12.2.3 Parse descriptor

The parse descriptor for a **`Ptry`** is the same as the parse descriptor for the underlying type.

12.2.4 Operations

The operations generated by the PADS compiler for a **`Ptry`** are those described in Chapter 3.

Read function

The error codes for **`Ptry`** are the same as for the underlying type.

Accumulator functions

Accumulator functions for **`Ptry`** are just as the accumulator functions for the underlying type.

Histogram functions

Histogram functions for **`Ptry`** are just as the histogram functions for the underlying type.

Clustering functions

Clustering functions for **`Ptry`** are just as the clustering functions for the underlying type.

Chapter 13

Precurs

Precur supports the description of recursive (*i.e.*, tree-structured) data.

13.1 Syntax

The syntax for **Precur** is given by the following BNF grammar fragment:

recur_ty ::= **Precur** [(*p_ty*)] identifier [*p_formals*] ;
| **Precur** *struct_ty*
| **Precur** *union_ty*

We explain the meaning of this syntax in the remainder of this chapter. All non-terminals not defined in this grammar fragment were defined previously: PADS types (*p_ty*) and parameter lists (*p_formals*) in Section 3.6, *struct_ty* in Section 5.1, and *union_ty* in Section 6.1.

13.1.1 Examples

The **Precur** construct has two components: a (forward) declaration and a definition. The forward declaration allows later types to refer to the recursive type before its definition, just as in C. The definition specifies the structure of the recursive data structure. Currently, the definition can be either a **Pstruct** or **Punion**.

We begin this example with a simple data fragment encoding a sorted binary tree containing the numbers 3,4,7,9, and 10.

```
(3,((4,(7,9)),10))
```

In this encoding (based on the Newick format [New]), a leaf is an integer while a full tree is a matched pair of parentheses enclosing two child trees separated by a comma. This encoding is naturally described recursively, as shown in the simple format below:

```
Precur tree;

Pstruct fullTree{
  '(';
  tree left;
  ',';
  tree right;
  ')';
};

Precur Punion tree{
  Pint32     value;
  fullTree   nested;
};
```

In this example, types `tree` and `fullTree` are mutually recursive. As recursive types are treated differently than other types, we must explicitly designate at least one of them as a recursive type. In this example, we choose `tree` to be the recursive type and so begin with a forward declaration of `tree` so that it may be referenced in `fullTree`.

13.2 Generated library

Precurs are different than other types in two important ways. First, in C, mutual recursion (of both functions and types) is handled through forward declarations. Therefore, for every recursive type, the compiler generates type and function forward declarations in addition to the standard type and function definitions.

Second, in general, we can't know the size of a data source described with recursive types. Therefore, the data structures corresponding to recursive types must be dynamically allocated. To accomplish this, the compiler generates two types for each of the generated data structures. One is an anonymous [1] type that is generated from the recursive type's definition. The other is a named type (where the name corresponds to the name of the recursive type) that is a pointer to the anonymous type.

We will use the `tree` examples to illustrate the data structures and functions generated for recursive types.

13.2.1 In-memory representation

The in-memory representation of a recursive type has two parts: the representation of the underlying type and a pointer to that representation. The name of the underlying type is derived from the name of the type by prefixing the name with an underscore ('_'). In our example, type `tree` is defined as follows:

```
typedef struct _tree_s *tree;
```

The definition of `struct _tree_s` is just the representation of the underlying **Punion**.

13.2.2 Mask

The mask of a recursive type has two parts: the mask of the underlying type and a pointer to that mask. The name of the underlying type is derived from the name of the type by prefixing the name with an underscore ('_'). In our example, type `tree_m` is defined as follows:

[1] For technical reasons, we do not actually generate an anonymous type, but rather generate for the type a fresh name that is not chosen by the user.

```
typedef struct _tree_m_s *tree_m;
```

The definition of `struct _tree_m_s` is just the mask of the underlying **Punion**. Note the difference from **Ptypedef**s, which provide a new mask type for the new type.

13.2.3 Parse descriptor

The parse descriptor of a **Precur** with name `myRecur` is a C struct with name `myRecur_pd`. This struct has the fields described in Section 3.13. In addition, it contains a field `val`, the type of which is a pointer to the parse descriptor for the underlying type.

For example, the parse descriptor type `tree_pd` has the following structure:

```
typedef struct tree_pd_s tree_pd;

struct tree_pd_s {
  Pflags_t pstate;
  Puint32 nerr;
  PerrCode_t errCode;
  Ploc_t loc;
  struct _tree_pd_s *val;
};
```

The definition of `struct _tree_pd_s` is just the parse descriptor of the underlying **Punion**.

13.2.4 Operations

The operations generated by the PADS compiler for a recursive type are those described in Chapter 3. However, the compiler generates two sets of functions for every recursive type. One is the standard set of functions for the underlying type. Note that these are not intended for use by the user. The other set of functions are those for operating on the type itself. These functions essentially wrap the operations of the underlying type and handle the dynamic allocation and deallocation of the representation and parse descriptor.

The operations appear in Figure 13.1.

Note that the mask is handled differently from the parse descriptor and representation. As the mask is an input to the parsing process, it must be fully allocated and initialized before parsing begins. However, due to the fact the size of the data is not known before parsing it, the mask must able to guide the parsing process for data of arbitrary size. The simplest way to accomplish this is to set the (recursive) pointers in the mask to point back to the root of the mask. In this way, we make the mask a cyclic data structure thereby simulating an infinitely large data structure. However, as traversal of the mask is driven by the data itself, this structure does not introduce a danger of infinite loops.

In the current release there is no support for automatically initializing masks to be cyclic. Instead, users must initialize the pointers correctly themselves. To assist, we provide a macro that bundles the various steps in initializing the mask. It allocates the mask, initializes it and then sets the specified pointer field to point back the root of the mask. It has the following signature:

```
P_DynamicMaskInit(m, maskTy, baseMaskTy, maskVal, PATH_TO_MASK)
```

The argument `m` is the (pointer) variable for the root of the mask. Argument `maskTy` is the type of the mask itself, and `baseMaskTy` is the underlying mask type. Argument `maskVal` specifies the default setting for all of the mask's fields. Finally, `PATH_TO_MASK`, specifies the path from the root of the mask to

```
Perror_t tree_init (P_t *,tree *);

Perror_t tree_pd_init (P_t *,tree_pd *);

Perror_t tree_cleanup (P_t *,tree *);

Perror_t tree_pd_cleanup (P_t *,tree_pd *);

Perror_t tree_copy (P_t *,tree *,tree *);

Perror_t tree_pd_copy (P_t *,tree_pd *,tree_pd *);

void tree_m_init (P_t *,tree_m *,Pbase_m);

Perror_t tree_read (P_t *,tree_m *,tree_pd *,tree *);

int tree_verify (tree *);

int tree_genPD (P_t *,tree *,tree_pd *);

ssize_t tree_write2buf (P_t *,Pbyte *,size_t,int *,tree_pd *,tree *);

ssize_t tree_write2io (P_t *,Sfio_t *,tree_pd *,tree *);

ssize_t tree_write_xml_2buf (P_t *,Pbyte *,size_t,int *,tree_pd *,tree *,char const *,int);

ssize_t tree_write_xml_2io (P_t *,Sfio_t *,tree_pd *,tree *,char const *,int);
```

Figure 13.1: Prototypes of operations generated for the **Precur** `tree`.

the pointer field in the mask. Note that this assumes that there is only one pointer field. In the case of more, the user must initialize those herself. Here is an example use of this macro for our `tree` format:

```
P_DynamicMaskInit(m, tree_m, _tree_m, READ_MASK, m->nested.left);
m->nested.right = m;
```

Note the need to initialize the second pointer separately.

Code such as this can be specified for some of the template programs with the macro `CUSTOM_MASK_CODE`.

Read function

The error codes for **Precur**s are:

Code	Meaning
`P_OK`	Indicates no error occurred
`P_ERR`	Indicates an unspecified error occured.

Accumulator functions

Accumulator functions for recursive types are not yet supported.

Histogram functions

Histogram functions for recursive types are not yet supported.

Clustering functions

Clustering functions for recursive types are not yet supported.

Chapter 14

Using the generated library

In this chapter, we describe various functions provided by the core PADS library for manipulating PADS handles. Figure 2.6 shows a simple example of library use: the call to `P_open` initializes the PADS handle, in this case with a default PADS discipline and a default IO discipline; the call to `P_io_fopen` indicates that the input data may be found in the file located at path `"data/sirius"`; the function `P_io_at_eof` tests if the file has been exhausted; the call to `P_io_close` closes the input file; and finally, the call to `P_close` closes the PADS handle.

`Perror_t P_open (P_t **pads_out, Pdisc_t *disc, Pio_disc_t *io_disc)` This function creates a PADS handle, which is returned in the space supplied by the first parameter. The second and third arguments are input parameters pointing to a PADS discipline and an IO discipline to use in the handle. Either or both of these may be `NULL`, in which case default disciplines are used.

`Perror_t P_close (P_t *pads)` This function deallocates a PADS handle, freeing all associated resources. If there is an installed IO discipline, it is unmade; after this point it should NOT be used any more.

`Perror_t P_close_keep_io_disc(P_t *pads, int keep_io_disc)` This function is Like `P_close`, except takes an extra argument, `keep_io_disc`, which if non-zero indicates the installed IO discipline (if any) should not be unmade; in this case it CAN be used again, , in a future `P_open` call.

`Pdisc_t * P_get_disc(P_t *pads)` This function returns `NULL` on error; otherwise, it returns a pointer to the installed discipline.

`Perror_t P_set_disc(P_t *pads, Pdisc_t *new_disc, int xfer_io)` This function installs a different discipline handle. If the parameter `xfer_io` is non-zero, then the IO discipline from the old handle is moved to the new handle.

`Perror_t P_set_io_disc(P_t* pads, Pio_disc_t* new_io_disc)` This function installs a different IO discipline into the main discipline. If there is an open SFIO stream, it is transferred to the new IO discipline after closing the old IO discipline in a way that returns all bytes beyond the current IO cursor to the stream. The old IO discipline (if any) is unmade. After this point the old IO discipine should NOT be re-used.

`Perror_t P_set_io_disc_keep_old(P_t* pads, Pio_disc_t* new_io_disc, int keep_old)` This function is like `P_set_io_disc`, except it takes an extra argument, `keep_old`, which if non-zero indicates that the old IO discipline should not be unmade; in this case it CAN be used again, *e.g.*, in a future `P_set_io_disc` call.

`Tm_zone_t *P_cstr2timezone(const char *tzone_str)` Utility function for converting a C string to a time zone pointer. It returns `NULL` if the string is an invalid time zone string. The following table describes the time zone names understood by the `P_cstr2timezone` function (columns two and three), as well as the numeric representations of such (columns four and five). Blank entries are represented by a dash.

Country	Standard	Savings	Minutes West of UTC	Saving Time Minutes Offset
-	GMT	-	0	0
-	UCT	-	0	0
-	UTC	-	0	0
-	CUT	-	0	0
USA	HST	-	600	0
USA	YST	YDT	540	-60
USA	PST	PDT	480	-60
USA	PST	PPET	480	-60
USA	MST	MDT	420	-60
USA	CST	CDT	360	-60
USA	EST	EDT	300	-60
CAN	AST	ADT	240	-60
CAN	NST	-	210	0
GBR	-	BST	0	-60
EUR	WET	-	0	-60
EUR	CET	-	-60	-60
EUR	MET	-	-60	-60
EUR	EET	-	-120	-60
ISR	IST	IDT	-180	-60
IND	IST	-	-330	0
CHN	HKT	-	-480	0
KOR	KST	KDT	-480	-60
SNG	SST	-	-480	0
JPN	JST	-	-540	0
AUS	AWST	-	-480	0
AUS	WST	-	-480	0
AUS	ACST	-	-570	-60
AUS	CST	-	-570	-60
AUS	AEST	-	-600	-60
AUS	EST	-	-600	-60
NZL	NZST	NZDT	-720	-60

`Perror_t P_set_in_time_zone(P_t *pads, const char *new_in_time_zone)`

`Perror_t P_set_out_time_zone(P_t *pads, const char *new_out_time_zone)` These functions set the input and output time zones, respectively. See Section 15.1.10 and Section 15.1.11 for more information.

`Perror_t P_io_set (P_t *pads, Sfio_t *io)` This function initializes or changes the current SFIO stream used for input. If there is already an installed SFIO stream, `P_io_close` is implicitly called first.

`Perror_t P_io_fopen(P_t *pads, const char *path)` This function opens a file for reading (a higher-level alternative to `io_set`). It uses `pads->disc->fopen_fn` if that value is non-null; otherwise, it uses `P_fopen`. It always opens files with mode `"r"`. The function returns `P_OK` on success, and `P_ERR` on error.

`Perror_t P_io_close(P_t *pads)` This function cleans up the IO discipline state. It attempts to return bytes that were read from the underlying SFIO stream but not consumed by the parse back to the stream.

If the underlying SFIO stream arose from a file open via `P_io_fopen`, the file is closed. If the underlying Sfio stream was installed via `P_io_set`, it is not closed. In this case, it is up to the program that opened the installed SFIO stream to close it (*after* calling `P_io_close`).

`Perror_t P_io_next_rec (P_t *pads, size_t *skipped_bytes_out)` This function advances the current IO position to start of the next record, if any. It returns `P_OK` on success, `P_ERR` on failure, which includes hitting `EOF` before `EOR`. For the `P_OK` case, the function sets `*skipped_bytes_out` to the number of data bytes that were passed over while searching for `EOR`.

`Perror_t P_io_skip_bytes(P_t *pads, size_t width, size_t *skipped_bytes_out)` This function advances the current IO position by specified number of bytes, or if that many bytes cannot be skipped, then by as many bytes as available. It sets `*bytes_skipped_out` to the number of bytes skipped. It returns `P_OK` if the requested bytes were skipped, `P_ERR` if fewer than the requested bytes were skipped. For record-based disciplines, the function does NOT advance the IO position beyond the current record.

`int` `P_io_at_eor(P_t *pads)` This function returns `1` if the current IO position is at `EOR`; otherwise it returns `0`.

`int` `P_io_at_eof(P_t *pads)` This function returns `1` if the current IO position is at `EOF`; otherwise it returns `0`.

`int` `P_io_at_eor_or_eof(P_t *pads)` This function returns `1` if the current IO position is at `EOR` or `EOF`; otherwise it returns `0`.

`const char` `* P_io_read_unit(P_t *pads)` This function provides a description of the read unit used in `Ppos_t` (*e.g.*, `"line"`, `"1K block"`, *etc.*). Returns `NULL` on error (if there is no installed IO discipline).

`Perror_t P_io_getPos(P_t *pads, Ppos_t *pos,` **`int`** `offset)` This function fills in `*pos` with the current IO position. If `offset` is zero, the current IO position is used, otherwise the position used is `offset` bytes from the current IO position. The current IO position does not change. `P_ERR` is returned if information about the specified position cannot be determined. `EOR` marker bytes (if any) are ignored when moving forward or back based on `offset`: `offset` only refers to data bytes.

`Perror_t P_io_getLocB(P_t *pads, Ploc_t *loc,` **`int`** `offset)` This function fills in `loc->b` with the IO position. See the description of `P_io_getPos` for a description of the `offset` parameter.

`Perror_t P_io_getLocE(P_t *pads, Ploc_t *loc,` **`int`** `offset)` This function fills in `loc->e` with the IO position. See the description of `P_io_getPos` for a description of the `offset` parameter.

`Perror_t P_io_getLoc(P_t *pads, Ploc_t *loc,` **`int`** `offset)` This function fills in both `loc->b` and `loc->e` with the IO position. See the description of `P_io_getPos` for a description of the `offset` parameter.

14.1 Compiled regular expressions

The scan and read functions that take regular expressions as arguments require pointers to compiled regular expressions of type `Pregexp_t*`.

A Pregexp_t contains two things:

1. a boolean, valid, which indicates whether the Pregexp_t contains a valid compiled regular expression.

2. some private state (an internal represention of the compiled regular expression) which should be ignored by the users of the library.

```
typedef struct Pregexp_s {
  int                   valid;
  P_REGEXP_T_PRIVATE_STATE;
} Pregexp_t;
```

If my_regexp.valid is non-zero, then my_regexp requires cleanup when no longer needed.

Upon declaring a Pregexp_t, one should set valid to 0. You can do this directly, as in:

```
Pregexp_t my_regexp =  0 ;
```

or you can use the preferred method, which is to use the following macro:

```
P_REGEXP_DECL_NULL(my_regexp);
```

When through with a Pregexp_t, one should call Pregexp_cleanup, as in:

```
Pregexp_cleanup(pads, &my_regexp);
```

to clean up any private state that may have been allocated.

The following functions are used to compile a string into a Pregexp_t and to cleanup a Pregexp_t when it is no longer needed. They should passed a pointer to a properly initialized (null or valid) Pregexp_t.

`Perror_t Pregexp_compile(P_t *pads, const Pstring *regexp_str, Pregexp_t *regexp)`
If regexp_str is a string containing a valid regular expression, this function fills in (*regexp) and returns P_OK. If the string is not a valid regular expression, it returns P_ERR.

`Perror_t Pregexp_compile_cstr(P_t *pads, const char *regexp_str, Pregexp_t *regexp)`
This function is like Pregexp_compile, but it takes a **const char*** argument rather than a **const** Pstring* argument.

`Perror_t Pregexp_cleanup(P_t *pads, Pregexp_t *regexp)` This function deallocates resources associated with regexp.

Both compile functions will perform a cleanup action if regexp->valid is non-zero prior to doing the compilation, and they both set regexp->valid to 0 if the compilation fails and to 1 if it succeeds.

Note that if you use a Pregexp_t to hold more than one compiled regular expression over time, you only need to call Pregexp_cleanup after the final use.

14.1.1 Regular expression macros

The `P_RE_STRING_FROM` macros convert their **`char`** or string args into strings containing regular expressions that match exactly the specified character or string. The string result is in temporary storage, so it should be used immediately (*e.g.*., in a `Pregexp_compile_cstr` call).

`const char``* P_RE_STRING_FROM_CHAR(P_t *pads, Pchar char_expr)` This function produces a regular expression string that matches a single character. **Example**: `P_RE_STRING_FROM_CHAR(pads, 'a')` returns string `"/[a]/"`.

`const char``* P_RE_STRING_FROM_CSTR(P_t *pads,` **`const char`** `* str_expr)` Produces a regular expression string that matches a string. **Example**: `P_RE_STRING_FROM_CSTR(pads, "abc")` returns string `"/abc/l"`.

`const char``* P_RE_STRING_FROM_STR(P_t *pads, Pstring *str_expr)` Same as above, but takes a `Pstring*` rather than a **`const char`**`*`.

`void` `P_REGEXP_FROM_CHAR(P_t *pads, Pregexp_t my_regexp, Pchar char_expr)`

`void` `P_REGEXP_FROM_CSTR(P_t *pads, Pregexp_t my_regexp,` **`const char`** `* str_expr)`

`void` `P_REGEXP_FROM_STR(P_t *pads, Pregexp_t my_regexp, Pstring * str_expr)` The `P_REGEXP_FROM` macros do the above conversions, and then do the added step of compiling the result into `Pregexp my_regexp`. In each case, one can check `my_regexp.valid` after the macro call to check whether the result is a valid compiled regular expression.

Chapter 15

Library customization

The PADS core library is parameterized by a main discpline and by an IO discipline that allow users to customize the behavior of the core system and the IO subsystem, respectively. One aspect of controlling IO is choosing what constitues a *Precord* in the data source.

15.1 The PADS discipline

A PADS handle, *i.e.*, a value of type `P_t`, contains a field named `my_disc` of type `Pdisc_t`, which appears in Figure 15.1. The various fields of `my_disc` allow users to customize aspects of the PADS system. We describe these fields in the following sections.

15.1.1 Version

This field stores the interface version of the PADS library. The C macro `P_VERSION` refers to the current version.

15.1.2 Control flags

The `flags` field is a combination of bit fields, described in the following sections. At the moment, only one such field is in use.

White space

If the `P_WSPACE_OK` bit is set in the `flags` field, then leading white space is allowed for variable-width ASCII integers. Similarly, leading and/or trailing white space is allowed for fixed-width ASCII integers.

15.1.3 Character encodings

PADS supports ASCII and EBCDIC encodings. The `def_charset` field indicates the character set for interpreting base types with no explicit character encoding by storing one of the following values:

```
Pcharset_ASCII
Pcharset_EBCDIC
```

```
typedef struct Pdisc_s {
  Pflags_t            version;        /* interface version */
  Pflags_t            flags;          /* control flags */
  Pcharset            def_charset;    /* default char set */
  int                 copy_strings;   /* if set,  ASCII string read
                                         functions copy strings */
  /* For the next four values, 0 means end-of-record /
     soft limit for non-record-based IO disciplines */
  size_t              match_max;      /* max match distance */
  size_t              numeric_max;    /* max numeric value distance */
  size_t              scan_max;       /* max normal scan distance */
  size_t              panic_max;      /* max panic scan distance */
  Pfopen_fn           fopen_fn;       /* file open function (default P_fopen) */
  Perror_fn           error_fn;       /* error function using  ... */
  PerrorRep           e_rep;          /* controls error reporting */
  Pendian_t           d_endian;       /* endian-ness of the data */
  Puint64             acc_max2track;  /* default maximum distinct values for
                                         accumulators to track */
  Puint64             acc_max2rep;    /* default maximum number of tracked values
                                         to describe in detail in report */
  Pfloat64            acc_pcnt2rep;   /* default maximum percent of values to
                                         describe in detail in report */
  const char         *in_time_zone;   /* default time zone for time input,
                                         specified as a string */
  const char         *out_time_zone;  /* default time zone for time formatted
                                         output, specified as a string */
  Pin_formats_t       in_formats;     /* default input formats */
  Pout_formats_t      out_formats;    /* default output formats */
  Pinv_val_fn_map_t  *inv_val_fn_map; /* map types to inv_val_fn
                                         for write functions */
  Pfmt_fn_map_t      *fmt_fn_map;     /* map types to fmt functions */
  Pio_disc_t         *io_disc;        /* sub-discipline for controlling IO */
} Pdisc_t;
```

Figure 15.1: The `Pdisc_t` type, which allows users to customize the behavior of the PADS system.

15.1.4 Copying strings

If `copy_string` field in PADS discipline is non-zero, the string read functions copy the strings found into the supplied representation, otherwise they do not. Instead, the target `Pstring` points to memory managed by the current IO discipline. This sharing avoids copying and speeds programs that will never reference an old record after a new one is read in. Field `copy_strings` should only be set to zero for record-based IO disciplines where strings from record K are not used after `P_io_next_rec` has been called to move the IO cursor to record K+1. Note: `Pstring_preserve` can be used to force a string that is using sharing to make a copy so that the string is 'preserved' (remains valid) across calls to `P_io_next_rec`.

15.1.5 Scanning extent

When scanning for a literal or regular expression match, how far should the scan/match go before giving up? If a record-based IO discipline is used, scanning and matching is limited to the scope of a single record. In addition, the following four `Pdisc_t` fields can be used to provide further constraints on scan/match scope.

`match_max` This field specifies the maximum number of bytes that will be included in an inclusive pattern match attempt (see, *e.g.* data type `Pstring_ME`). If set to zero, no `match_max` constraint is imposed

for a record-based IO discipline (other than finding end-of-record), whereas a built-in soft limit of P_BUILTIN_MATCH_MAX characters is imposed for non-record-based IO disciplnes. (The built-in limit is soft because if the match happens to get more than P_BUILTIN_MATCH_MAX characters in a single IO discipline read call it will go ahead and consider all of them. In contrast, if the discipline match_max is set explicitly to value K, then this is a hard limit: the match will only consider K characters even if more are available.)

numeric_max This field specifies the maximum number of bytes that will be included in an attempt to read a character-based representation of a number. If non-zero, the field should be set large enough to cover any leading white space (if allowed by P_WSPACE_OK), an optional +/- sign, and the digits (dot *etc.* for floats) that make up the numeric value. A value of zero for numeric_max results in an end-of-record constraint for record-based IO disciplines and in a soft limit of P_BUILTIN_NUMERIC_MAX bytes for non-record-based IO disciplines.

scan_max This field specifies the maximum number of bytes that will be considered by a normal scan that is looking for a terminating literal or a terminating regular expression (see, *e.g.* data type Pstring_SE.). Note that this includes both the bytes skipped plus the bytes used for the match. A scan_max of zero results in an end-of-record constraint for record-based IO disciplines and in a soft limit of P_BUILTIN_SCAN_MAX bytes for non-record-based IO disciplines.

panic_max This field specifies the maximum number of bytes that will be considered by when parsing hits a 'panic' state and is looking for a synchronizing literal or pattern. See, for example, termination conditions for user-defined array types. A panic_max of zero results in an end-of-record constraint for record-based IO disciplines and in a soft limit of P_BUILTIN_PANIC_MAX bytes for non-record-based IO disciplines.

For non-record-based IO disciplines, the default soft limits may be either too small or too large for a given input type. It is important for these cases to determine appropriate hard limit settings.

The built-in soft limits for use with non-record-based IO disciplines are as follows. Although you can change them and recompile the PADS library, it is easier to simply set up the correct hard limits in the discipline.

```
#define P_BUILTIN_MATCH_MAX       512
#define P_BUILTIN_SCAN_MAX        512
#define P_BUILTIN_NUMERIC_MAX     512
#define P_BUILTIN_PANIC_MAX      1024
```

15.1.6 File open function

The field fopen_fn stores the file open function used by P_io_fopen. If this field is NULL, the default function P_fopen is used.

A Pfopen_fn takes arguments (source, mode) and opens file source in the specified mode and returns an SFIO stream on success or NULL on failure.

```
typedef Sfio_t* (*Pfopen_fn)(const char *source, const char *mode);
```

It should normally have the the same semantics as the call sfopen(NiL, string, mode), except that for the string constants it should return an existing SFIO stream:

```
"/dev/stdin"     ⟶   sfstdin
"/dev/stdout"    ⟶   sfstdout
"/dev/stderr"    ⟶   sfstderr
```

For "/dev/stdin", mode "r" (read) is expected. For "/dev/stdout or "/dev/stderr", mode "a" (append-only) is expected. If you use some other mode for these three cases, the function should attempt to apply mode to the specified SFIO stream; it should return NULL if this fails, otherwise the specified stream.

15.1.7 Error function

The field error_fn stores the error reporting function. If this field is NULL, the default function P_error is used. The type for this function is:

```
typedef int (*Perror_fn)(const char *libnm, int level, ...);
```

Error functions are a lot like the standard printf funtion, with two additional arguments, libnm and level. For example, where one might use printf as follows:

```
printf("count: %d\n", 10);
```

one can do the same thing with P_error using:

```
P_error(NULL, P_LEV_INFO, "count: %d", 10);
```

Note that P_error automatically adds a newline, so we did not have to include a "\n" in the format string, as we did with the printf example.

The first argument to an error function, libnm, is normally NULL (it is for the name of the library calling the error function).

The second argument, level, is used to specify the severity of the error. Level P_LEV_INFO is used for an informative (non-error) message, P_LEV_WARN is used for a warning, P_LEV_ERR for a non-fatal error, and P_LEV_FATAL for a fatal error. One can 'or' in the following flags (as in P_LEV_WARN|P_FLG_PROMPT):

P_FLG_PROMPT	do not emit a newline
P_FLG_SYSERR	add a description of errno (errno should be a system error)
P_FLG_LIBRARY	error is from library

Library messages are forced if the environment variable ERROR_OPTIONS includes the string "library"; SYSERR (errno) messages are forced if it includes the string "system."

For convenience, if the first argument, library name libnm, is non-NULL, then flag P_FLG_LIBRARY is automatically or'd into level.

15.1.8 Endian-ness

The field d_endian stores the endianness of the data. It can have the value PbigEndian or PlittleEndian. If d_endian does not equal the endiannes of the machine running the parsing code, the bye order of binary integeeres is swapped by the binary integer read function.

15.1.9 Accumulator customization

The fields acc_max2track, acc_max2rep, and acc_pct2rep allow the user to customize accumulator behavior. Section 16.2 describes these fields in detail.

15.1.10 Input time zone

The field in_time_zone specifies the default time zone for string-based time input, used for input date strings that do not have time zone information in them. For example, the date 15/Oct/1997:18:46:51 has no time zone information. If in_time_zone is set to "UTC", then this date/time would be assumed to be a UTC time. In contrast, regardless of the in_time_zone setting, the date 15/Oct/1997:18:46:51 -0700 will always be interpreted as being in a time zone seven hours earlier than UTC time.

The in_time_zone is passed to the tmzone function, so it must be a time zone description that tmzone understands. Intuitively, it accepts three-letter strings such as "PST" and "EDT" as well as strings denoting numeric offsets from UTC time, such as "-0500". Chapter 14 describes the legal time zone designation strings. Documentation for the tmzone function appears on the web page: www.research.att.com/~gsf/man/man3/tm.html

Before calling P_open, the discipline field disc->in_time can be initialized directly. After calling P_open, however, it must be changed by passing the pads handle and a time zone string to P_set_in_time_zone, *e.g.*,

```
P_set_in_time_zone(pads,"PST");
```

This will set pads->disc->in_time_zone, and will also update an internal representation of the time zone maintained as part of the pads state.

15.1.11 Output time zone

This field specifies the output time zone for formatted time output. Regardless of the time zone used to read in a time, disc->output_time_zone controls which time zone is used when formatting the time for output. For example, a time that is read as 6am UTC time would be formatted as 1am if the output_time_zone is "-0500". Note that in the normal case you should use the same time zone for both input and output, unless you are intentially translating times from one time zone to another one. The format of output time zone specification strings is the same as for input time zone. Chapter 14 describes the legal time zone designation strings.

Before calling P_open, the discipline field disc->in_time can be initialized directly. After calling P_open, however, it must be changed by passing the pads handle and a time zone string to P_set_output_time_zone, *e.g.*,

```
P_set_output_time_zone(pads,"CDT");
```

This will set pads->disc->output_time_zone, and will also update an internal representation of the output time zone maintained as part of the pads state.

15.1.12 Input formats

The in_formats field of the discipline allows one to specify default input formats for some special types where there is in no 'obvious' default. Figure 15.2 contains the type of this field. The current entries are:

in_formats.timestamp This field contains a format string specifying the input timestamp format, for use with Ptimestamp and its variants. Alternatives can be given using %|, and the special %& format can be used to indicate all formats that can be parsed by the tmdate fuction. The default,

```
"%m%d%y+%H%M%S%|%m%d%y+%H%M%S%|%m%d%Y+%H%M%S%|%m%d%Y+%H%M%S%|%&"
```

```
typedef struct Pin_formats_s {
  const char        *timestamp;
  const char        *date;
  const char        *time;
} Pin_formats_t;
```

Figure 15.2: The Pin_formats_t type, which allows users to specify the input format of various PADS base types. Each of the fields must be a non-null string with a format understood by the tmdate function.

allows for timestamps of these forms:

```
091172+230202
091172+23:02:02
09111972+230202
09111972+23:02:02
```

and also allows for all date/time formats parsed by tmdate. Documentation for the tmdate function appears on the web page: www.research.att.com/~gsf/man/man3/tm.html

in_formats.date A format string specifying the input date format, for use with Pdate and its variants. The default,

```
"%m%d%y%|%m%d%Y%|%&"
```

allows for dates of these two forms:

```
091172
09111972
```

and also allows for all date formats parsed by tmdate. Documentation for the tmdate function appears on the web page: www.research.att.com/~gsf/man/man3/tm.html

in_formats.tme A format string specifying the input time format, for use with Ptime and its variants. The default,

```
"%H%M%S%|%H:%M:%S%|%&"
```

allows for times of these two forms:

```
230202
23:02:02
```

and also allows for all date formats parsed by tmdate. Documentation for the tmdate function appears on the web page: www.research.att.com/~gsf/man/man3/tm.html

15.1.13 Output formats

The out_formats field of the discipline allows one to specify default output formats for some special types where there is in no 'obvious' default. Figure 15.3 contains the type of this field. The current entries are:

out_formats.timestamp

```
typedef struct Pout_formats_s {
  const char        *timestamp_explicit;
  const char        *timestamp;
  const char        *date_explicit;
  const char        *date;
  const char        *time_explicit;
  const char        *time;
} Pout_formats_t;
```

Figure 15.3: The `Pout_formats_t` type, which allows users to specify the output format of various PADS base types. Each of the fields must be a non-null string with a format understood by the `fmttime` function.

`out_formats.timestamp_explicit` These two values specifying the default output formats for the `Ptimestamp` and `Ptimestamp_explicit` families of types, respectively. The normal use is for these formats to describe both the date and time of day. Some examples:

```
"%Y%m%d|%H%M%S"
"%m/%d/%Y %H:%M"
"%K" /*default -- %K is the same as "%Y-%m-%d+%H:%M:%S"*/
```

Documentation for the `tmdate` function appears on the web page: `www.research.att.com/~gsf/man/man3/tm.html`

`out_formats.date`

`out_formats.date_explicit` These two values specifying the default output formats for the `Pdate` and `Pdate_explicit` families of types, respectively. The normal use is for these formats to describe the date but not the time of day. Some examples:

```
"%Y%m%d"
"%m/%d/%Y"
"%Y-%m-%d" /* default */
```

Documentation for the `tmdate` function appears on the web page: `www.research.att.com/~gsf/man/man3/tm.html`

`out_formats.time`

`out_formats.time_explicit` These two values specifying the default output formats for the `Ptime` and `Ptime_explicit` families of types, respectively. The normal use is for these formats to describe a time of day but not the date. Some examples:

```
"%H%M%S"
"%H.%M"
"%H:%M:%S" /* default */
```

Documentation for the `tmdate` function appears on the web page: `www.research.att.com/~gsf/man/man3/tm.html`

15.1.14 Writing invalid values

Write functions take a parse descriptor and a value. The value is valid if the parse descriptor's `errCode` is set to `P_NO_ERR`. The value has been filled in if the `errCode` is `P_USER_CONSTRAINT_VIOLATION`. For other `errCodes`, the value should be assumed to contain garbage. For invalid values, the write function must still write SOME value. For every type, one can specify an `inv_val` helper function that produces an invalid value for the type, to be used by the type's write functions. If no function is specified, then a default

invalid value is used, where there are two cases: if the errCode is P_USER_CONSTRAINT_VIOLATION, then the current invalid value is used; for any other errCode, a default invalid value is used.

The map from write functions to inv_val functions is found in the discipline in the field inv_val_fn_map, which maps values of type const char * (the string form of the type name) to Pinv_val_fn functions. If the inv_val_fn_map field is NULL, no mappings are used.

An invalid value function that handles type T values always takes four arguments:

- The P_t* handle
- A pointer to a type T parse descriptor
- A pointer to the invalid type T representation
- A va_list argument that encodes the extra type parameters (if any) associated with the type. For example, type Pa_int32_FW(:<width>:) has a single type parameter (width) of type Puint32, so va_list would encode a single Puint32 argument.

Arguments two and three use void* types to enable the table to be used with arbitrary types, including user-defined types. One must cast these void* args to the appropriate pointer types before use; see the example below. The function should replace the invalid value with a new value, and return P_OK on success and P_ERR if a replacement value has not been set.

Use P_set_inv_val_fn to set a function pointer, P_get_inv_val_fn to do a lookup:

```
Pinv_val_fn P_get_inv_val_fn(P_t* pads, Pinv_val_fn_map_t *map,
                             const char *type_name);
Pinv_val_fn P_set_inv_val_fn(P_t* pads, Pinv_val_fn_map_t *map,
                             const char *type_name, Pinv_val_fn fn);
```

Example: suppose an Pa_int32 field has an attached constraint that requires the value must be at least negative thirty. If a value of negative fifty is read, errCode will be P_USER_CONSTRAINT_VIOLATION. If no inv_val function has been provided, then the Pa_int32 write function will output -50. If the read function fails to read even a valid integer, the errCode will be P_INVALID_A_NUM, and the Pa_int32 write function will output P_MIN_INT32 (the default invalid value for all int32 write functions). If one wanted to correct all user contraint cases to use value -30, and to use P_MAX_INT32 for other invalid cases, one could provide an inv_val helper function to do so:

```
Perror_t my_int32_inv_val(P_t *pads, void *pd_void, void *val_void, va_list type_args) {
  Pbase_pd *pd  = (Pbase_pd*)pd_void;
  Pint32   *val = (Pint32*)val_void;
  if (pd->errCode == P_USER_CONSTRAINT_VIOLATION) {
     (*val) = -30;
  } else {
     (*val) = P_MAX_INT32;
  }
  return P_OK;
};

/*create call only needed if no map installed yet*/
pads->disc->inv_val_fn_map = Pinv_val_fn_map_create(pads);
P_set_inv_val_fn(pads, pads->disc->inv_val_fn_map, "Pint32", my_int32_inv_val);
```

Note that for a type T with three forms, P_T, Pa_T, and Pe_T, there is only one entry in the inv_val_fn_map, under string "P_T". For example, use "Pint32" to specify an invalid value function for all of these types: Pint32, Pa_int32, and Pe_int32.

An inv_val_fn for a string type should use Pstring_copy, Pstring_cstr_copy, Pstring_share, or Pstring_cstr_share to fill in the value of the Pstring* param.

15.2 The IO Discpline

IO discipline values, which have type `Pio_disc_t`, control the 'raw' reading of data from a file or from some other data source. The PADS system provides a collection of functions for generating various IO disciplines, corresponding to various kinds of record structures: new-line terminated, fixed width, IBM-style (initial data indicating size of record, followed by payload), *etc.* In addition, the discipline indicates if the data source is seekable (a file) or not (a stream).

To use an IO discipline, the user first creates one by invoking a creation function supplied by the PADS system. The resulting IO discipline is then installed by passing it as an argument to either `P_open` or to `P_set_io_disc`.

`Pio_disc_t * P_fwrec_make(size_t leader_len, size_t data_len, size_t trailer_len)`
Instantiates an instance of `fwrec`, a discipline for fixed-width records. The parameter `data_len` specifies the number of data bytes per record, while `leader_len` and `trailer_len` specify the number of bytes that occur before and after the data bytes within each record (either or both can be zero). Thus the total record size in bytes is the sum of the three arguments.

`Pio_disc_t * P_fwrec_noseek_make(size_t leader_len, size_t data_len, size_t trailer_len)`
Instantiates an instance of `fwrec_noseek`, a version of `norec` that does not require that the SFIO stream is seekable.

`Pio_disc_t * P_ctrec_make(Pbyte termChar, size_t block_size_hint);` Instantiates an instance of `ctrec`, a discipline for character-terminated variable-width records. Argument `termChar` is the character that marks the end of a record. Argument `block_size_hint` suggests a block size to use, if the discipline chooses to do fixed block-sized reads 'under the covers'. It may be ignored by the discipline. For ASCII newline-terminated records use, `'\n'` or `P_ASCII_NEWLINE` as the term character. For `EBCDIC` newline-terminated records, use `P_EBCDIC_NEWLINE` as the term character.

`Pio_disc_t * P_ctrec_noseek_make(Pbyte termChar, size_t block_size_hint)` Instantiates an instance of `ctrec_noseek`, a version of `norec` that does not require that the SFIO stream is seekable.

`Pio_disc_t * P_nlrec_make(size_t block_size_hint)` Shorthand for calling the corresponding `ctrec` make function with `'\n'` as `termChar`.

`Pio_disc_t * P_nlrec_noseek_make(size_t block_size_hint)` Shorthand for calling the corresponding `ctrec` make function with `'\n'` as `termChar`.

`Pio_disc_t * P_vlrec_make(int blocked, size_t avg_rlen_hint)` Instantiates an instance of `vlrec`, a discipline for IBM-style variable-length records with record length specified at the start of each record. If blocked is set (`!= 0`) then the records are grouped into blocks, where each block has a length given at the start of each block. Argument `avg_rlen_hint` is a hint as to what the average record length is, to help the discipline allocate memory. It should include the four bytes at the start of each record used for the record length. It may be ignored by the discipline.

`Pio_disc_t * P_vlrec_noseek_make(int blocked, size_t avg_rlen_hint)` Instantiates an instance of `vlrec_noseek`, a version of `vlrec` that does not require that the SFIO stream is seekable.

`Pio_disc_t * P_norec_make(size_t block_size_hint)` Instantiates an instance of `norec`, a raw bytes discipline that does not use records. Argument `block_size_hint` is a hint as to what block size to use, if the discipline chooses to do fixed block-sized reads 'under the covers'. It may be ignored by the discipline.

`Pio_disc_t * P_norec_noseek_make(size_t block_size_hint)` Instantiates an instance of `norec_noseek`, a version of `norec` that does not require that the SFIO stream is seekable.

15.2.1 Closing an IO discipline

When an IO discipline is no longer needed, the user should unmake it. The function `P_io_disc_unmake` explicitly deallocates an IO discipline. In addition, the function `P_close` deallocates the installed IO discipline. The function `P_set_io_disc` deallocates the previously installed discipline. If desired, an IO discipline can be preserved using `P_close_keep_io_disc` or `P_set_io_disc_keep_old`, in which case it can be re-used in a future `P_open` or `P_set_io_disc` call.

15.2.2 Implementations

Implementations of the standard IO disciplines can be found in `libpads/default_io_disc.c`. Anyone planning to implement a new IO discipline should consult `default_io_disc.c`.

15.3 Adding new base types

Chapter 16

Accumulators

Accumulators summarize data inserted into them. They are useful for quickly computing a "bird's-eye" view of a given data source. For each piece of a PADS description, the accumulator summarizes the percentage of errors seen and reports the most frequently seen values. For example, when run on sample web server log data, the accumulator report for the length field contains the information shown in Figure 16.1. By default, accumulators track the first 1000 distinct values seen in the data source and report the frequency of the top ten values. In this particular run, 99.552% of all values were tracked.

16.1 Operations

Figure 16.2 shows the accumulator type declaration and associated functions for a PADS type. These functions have the following behaviors:

`entry_t_acc_init` Initializes accumulator data structure. This function must be called before any data can be added to the accumulator.

`entry_t_acc_reset` Reinitializes accumulator data structure, erasing all information previously stored.

`entry_t_acc_cleanup` Deallocates all memory associated with accumulator.

`entry_t_acc_add` Inserts argument in-memory representation and parse descriptor into argument accumulator. The parse descriptor allows the accumulator to track errors as well as legal values.

`entry_t_acc_report2io` Writes summary report for accumulator `acc` to open SFIO stream `outstr`. The argument `prefix` is a descriptive string, usually the path to the data being accumulated. If `NULL`, the string `"<top>"` is used. In the accumulator snippet in Figure 16.1, this path is `<top>.length`. The argument `what` is a string describing the kind of data. If `NULL`, a short for of the accumulator is used as a default, *e.g.* `uint32` for `Puint32`. The argument `nst` indicates the nesting level. Level zero should be used for a top-level call. Reporting routines bump the nesting level for recrsive report calls that describe sub-parts. Nesting level `-1` indicates a minimal prefix header should be output, *i.e.*, just the prefix without any adornment.

`entry_t_acc_report` Writes summary report for accumulator `acc` to standard error. The other arguments are the same as for `entry_t_acc_report2io`

Figure 16.3 illustrates a sample use of accumulator functions for printing a summary of CLF `entry_t`s.

```
<top>.length : uint32
++++++++++++++++++++++++++++++++++++++++
good: 53544   bad: 3824    pcnt-bad: 6.666
min: 35  max: 248591  avg: 4090.234
top 10 values out of 1000 distinct values:
tracked 99.552% of values
 val:  3082 count:  1254  %-of-good:  2.342
 val:   170 count:  1148  %-of-good:  2.144
 val:    43 count:  1018  %-of-good:  1.901
 val:  9372 count:   975  %-of-good:  1.821
 val:  1425 count:   896  %-of-good:  1.673
 val:   518 count:   893  %-of-good:  1.668
 val:  1082 count:   881  %-of-good:  1.645
 val:  1367 count:   874  %-of-good:  1.632
 val:  1027 count:   859  %-of-good:  1.604
 val:  1277 count:   857  %-of-good:  1.601
. . . . . . . . . . . . . . . . . . . . .
 SUMMING    count:  9655  %-of-good: 18.032
```

Figure 16.1: Portion of accumulator report for length field of web server log data.

16.2 Customization

The PADS discpline allows users to customize various aspects of accumulation by setting the appropriate field in the discpline. If `pads` is an active PADS handle, then `pads->disc` provides access to the discipline, which contains the following accumulator related fields:

`acc_max2track` is a `Puint64` denoting the default maximum number of distinct values for accumulators to track. Setting this field to `P_MAX_UINT64` indicates no limit. Note that the higher the value, the more memory accumulators will consume. By default, the PADS system sets this value to `1000`. When an `acc_init` function is called on a base-type accumulator `a`, the field `a.max2track` is set to `pads->disc->acc_max2track`. The value `a.max2track` may be modified by hand after this call to force the accumulator `a` to use a non-default value.

`acc_max2rep` is a `Puint64` denoting the default number of tracked values for accumulators to describe in detail in the generated report. Setting this field to `P_MAX_UINT64` indicates no limit on the tracked values to display. By default, the PADS system sets this value to ten. When an `acc_init` function is called on a base-type accumulator `a`, `a.max2rep` is set to `pads->disc->acc_max2rep`. The value `a.max2rep` can be modified by hand after this call to force the accumulator `a` to use a non-default value.

`acc_pcnt2rep` is a `Pfloat` denoting the default percent of values for accumulators to describe in detail in the generated report. Setting this field to `100.0` indicates no limit on the set of tracked values to display. By default, PADS sets this value to `100.0`. Upon calling an `acc_init` function on some base-type accumulator `a`, `a.pcnt2rep` is set to `pads->disc->acc_pcnt2rep`. `a.pcnt2rep` can be modified by hand after this call to force the accumulator `a` to use a non-default value.

Note that both `acc_max2rep` and `acc_pcnt2rep` set a limit on the number of tracked values to display. The reporting stops when either limit occurs.

Generated accumulators have components that are base-type accumulators. Thus, after initializing some generated accumulator `a`, one could modify `a.foo.bar.max2track` or `a.foo.bar.max2rep` to change the tracking or reporting of the `foo.bar` component `a`.

```
typedef struct  {
  Puint32_acc nerr;
  order_header_t_acc h;
  eventSeq_t_acc events;
} entry_t_acc;

Perror_t entry_t_acc_init (P_t *pads,entry_t_acc *acc);
Perror_t entry_t_acc_reset (P_t *pads,entry_t_acc *acc);
Perror_t entry_t_acc_cleanup (P_t *pads,entry_t_acc *acc);
Perror_t entry_t_acc_add (P_t *pads,entry_t_acc *acc,
                          entry_t_pd *pd,entry_t *rep);
Perror_t entry_t_acc_report2io (P_t *pads,Sfio_t *outstr,char const *prefix,
                                char const *what,int nst,entry_t_acc *acc);
Perror_t entry_t_acc_report (P_t *pads,char const *prefix,
                             char const *what,
                             int nst,entry_t_acc *acc);
```

Figure 16.2: Accumulator functions generated for the `entry_t` type.

16.3 Template Program

Because generating an accumulator report from a PADS description is a very routine task, PADS provides a template program to automate the task for common data formats. In particular, the template applies to data that can be viewed as an optional header followed by a sequence of records. Note that any data source that can be read entirely into memory fits this pattern by considering the source to have no header and a single body record.

When instantiated, the template program takes an optional command-line argument specifying the path to the data source. If no argument is given, it uses a default location for the data specified by the template user. The template first reads the optional header, then reads each record and inserts the resulting value into an accumulator until the data source is exhausted, at which point it prints the accumulator report to standard error. The code in Figure 2.7 illustrates using the accumulator template `template/accum_report.h`. This template is a C header file parameterized by a number of macros that permit the user to customize the template by defining appropriate values for these macros. For example, in the code in Figure 2.7, the user defines the macros `DEF_INPUT_FILE`, `PADS_TY`, and `IO_DISC_MK` to indicate the default input file, the type of the repeated record in the data source, and the IO discipline. The following list describes these and the other macros used by the accumulator template:

`DATE_IN_FMT` If defined, this macro sets the default input format for dates described by `Pdate`. See Section 15.1.12 for more information.

`DATE_OUT_FMT` If defined, this macro sets the default output format for `Pdate` and `Pdate_explicit`. See Section 15.1.13 for more information.

`DEF_INPUT_FILE` If defined, this macros specifies a string representation of the path to the default data source. If no path to the data is supplied at the command-line, this is the location used for input data.

`EXTRA_BAD_READ_CODE` If defined, this macro points to a C statement that will be executed after any body record containing an error.

`EXTRA_BEGIN_CODE` If defined, this macro points to a C statement that will be executed after all initialization code is performed, but before the optional header is read.

`EXTRA_DECLS` This optional macro defines additional C declarations that proceed all accumulator code.

```
#include "wsl.h"
#define DEF_INPUT_FILE  "data/wsl"

int main(int argc, char** argv) {
  P_t                 *pads;
  Pio_disc_t          *io_disc;
  entry_t             rep;
  entry_t_pd          pd;
  entry_t_m           mask;
  entry_t_acc         acc;
  char                *fname = DEF_INPUT_FILE;

  io_disc = P_nlrec_noseek_make(0);
  P_open(&pads, 0, io_disc);

  entry_t_init(pads, &rep);
  entry_t_pd_init(pads, &pd);
  entry_t_m_init(pads, &mask, P_CheckAndSet);

  if (P_ERR == P_io_fopen(pads, fname)) {
    error(2, "*** P_io_fopen failed ***");
    return -1;
  }

  entry_t_acc_init(pads, &acc);
  while (!P_io_at_eof(pads)) {
    entry_t_read(pads, &mask, &pd, &rep);
    entry_t_acc_add(pads, &acc, &pd, &rep);
  };
  entry_t_acc_report(pads, "", 0, 0, &acc);

  P_io_close(pads);
  entry_t_cleanup(pads, &rep);
  entry_t_pd_cleanup(pads, &pd);
  entry_t_acc_cleanup(pads, &acc);
  P_close(pads);
  return 0;
}
```

Figure 16.3: Simple use of accumulator functions for the `entry_t` type from CLF data.

`EXTRA_DONE_CODE` If defined, this macro points to a C statement that will be executed after generating the accumulator report.

`EXTRA_GOOD_READ_CODE` If defined, this macro points to a C statement that will be executed after any body record not containing an error.

`EXTRA_HEADER_READ_ARGS` If the type of the header record was parameterized, this macro allows the user to supply corresponding parameters.

`EXTRA_READ_ARGS` If the type of the repeated record was parameterized, this macro allows the user to supply corresponding parameters.

`IN_TIME_ZONE` If set, this macro specifies the input time zone of date types that do not include time zone information. See Section 15.1.10 for more detail.

`IO_DISC_MK` If defined, this macro specifies the interpretation of **`Precord`** by indicating which IO discpline the system should install. It specifies the discipline by naming the function to create the discipline. Section 15.2 describes the available IO discipline creation functions. If the user does not define this macro, the system installs the IO discipline corresponding to new-line terminated ASCII records.

`MAX_RECS` If defined, this macro specifies an integer that limits the number of repeated records that the accumulator program should read.

`OUT_TIME_ZONE` If set, this macro specifies the output time zone of date types. See Section 15.1.11 for more detail.

`PADS_HDR_TY` Intuitively, this macro defines the type of the header record in the data source. This macro need only be defined if the data source has a header record. It defines a function used by the template program to generate the various function and type names derived from the name of the header record type, *i.e.*, the type of the associated in-memory representation, mask, parse descriptor, read function, *etc.*

`PADS_TY` Intuitively, this macro defines the type of the repeated record in the data source, *i.e.*, the type of the value to be accumulated. This macro must be defined to use the accumulator template. It defines a function used by the template program to generate the various function and type names derived from the name of the record type, *i.e.*, the type of the associated in-memory representation, mask, parse descriptor, read function, *etc.*

`READ_MASK` This macro specifies the mask to use in reading the repeated record. If not defined by the user, the template uses the value `P_CheckAndSet`.

`TIME_IN_FMT` If defined, this macro sets the default input format for `Ptime`. See Section 15.1.12 for more information.

`TIME_OUT_FMT` If defined, this macro sets the default output format for `Ptime` and `Ptime_explicit`. See Section 15.1.13 for more information.

`TIMESTAMP_IN_FMT` If defined, this macro sets the default input format for `Ptimestamp`. See Section 15.1.12 for more information.

`TIMESTAMP_OUT_FMT` If defined, this macro sets the default output format for the PADS types `Ptimestamp` and `Ptimestamp_explicit`. See Section 15.1.13 for more information.

`WSPACE_OK` If defined, this macro indicates that leading white space for variable-width ASCII integers is okay, as well as leading and trailing white space for fixed-width ASCII integers.

Chapter 17

Histogram

A histogram is a piecewise-constant approximation of an observed data distribution. It is used as a small space, approximate synopsis of the underlying data distribution which are ofter too large to be stored precisely. Histograms are built for each meaningful piece of a PADS description. Figure 17.1 is an example report for the length field of a web server log data. In this particular run, optimal 5-bucket histogram is built for every 500 values seen in the data source.

17.1 Operations

Figure 17.2 shows the histogram functions declared for a PADS type. These functions have the following behaviors:

`entry_t_hist_init` Initializes histogram data structure. This function must be called before any data can be added to the histogram.

`entry_t_hist_setPara` Customizes histogram data structure. For the two conversion functions (specified below), user needs to set the corresponding fields in the template program. This function must be called to make any customization effected.

`entry_t_hist_reset` Reinitializes histogram data structure. This function can be used to set any point of the data source as the start point of a new run. But it can't be used to reset any previous defined parameters.

`entry_t_hist_cleanup` Deallocates all memory associated with histogram.

`entry_t_hist_add` Inserts a data value. This function is called once a new record is coming. Any data type with an associated mapping function to `Pfloat64` is considered as a meaningful type. This function tracks fields with meaningful type and legal values only. The output parameter isFull will be set nonzero, if the current data is the last one of a portion.

`entry_t_hist_reportFull2io` Writes summary report for finished histograms to `*outstr`. In most cases, when this function is called, all stored histograms will be reported and the space will be released, while the current one won't.

`entry_t_hist_reportAll2io` Writes summary report for all histograms to `*outstr`. When this function is called, all histograms will be reported and the space will be released.

`entry_t_hist_reportFull` Writes summary report for finished histograms to screen.

```
*** Histogram Result ***
From 0 to 397, with height 4016.
From 398 to 398, with height 36122.
From 399 to 423, with height 5584.
From 424 to 424, with height 33250.
From 425 to 499, with height 3126.
*** Histogram Result ***
From 0 to 196, with height 3286.
From 197 to 197, with height 30430.
From 198 to 313, with height 3233.
From 314 to 314, with height 30430.
From 315 to 499, with height 3655.
*** Histogram Result ***
From 0 to 242, with height 3686.
From 243 to 243, with height 36122.
From 244 to 441, with height 3720.
From 442 to 442, with height 26074.
From 443 to 499, with height 7035.
*** Histogram Result ***
From 0 to 93, with height 4204.
From 94 to 94, with height 36122.
From 95 to 206, with height 3000.
From 207 to 210, with height 21496.
From 211 to 499, with height 3890.
. . . . . .
```

Figure 17.1: Portion of histogram report for length field of web server log data.

```
Perror_t entry_t_hist_init (P_t *pads,entry_t_hist *h);
Perror_t entry_t_hist_setPara (P_t *pads,entry_t_hist *h,P_hist *d_hist);
Perror_t entry_t_hist_reset (P_t *pads,entry_t_hist *h);
Perror_t entry_t_hist_cleanup (P_t *pads,entry_t_hist *h);
Perror_t entry_t_hist_add (P_t *pads,entry_t_hist *h,Pbase_pd
*pd,entry_t *rep,Puint32 *isFull);
Perror_t entry_t_hist_reportFull2io (P_t *pads, Sfio_t *ourstr,
const char *prefix, const char *what, int nst, entry_t_hist *h);
Perror_t entry_t_hist_reportAll2io (P_t *pads, Sfio_t *ourstr,
const char *prefix, const char *what, int nst, entry_t_hist *h);
Perror_t entry_t_hist_reportFull (P_t *pads, const char *prefix,
const char *what, int nst, entry_t_hist *h);
Perror_t entry_t_hist_reportAll (P_t *pads, const char *prefix,
const char *what, int nst, entry_t_hist *h);
```

Figure 17.2: Histogram functions generated for the `entry_t` type.

```
#include "wsl.h"
#define DEF_INPUT_FILE  "data/wsl"

int main(int argc, char** argv) {
  P_t                   *pads;
  Pio_disc_t           *io_disc;
  P_hist                default_hist;
  entry_t               rep;
  entry_t_pd           pd;
  entry_t_m            mask;
  entry_t_hist         h;
  Puint32                isFull;
  char              *fname = DEF_INPUT_FILE;

  io_disc = P_nlrec_noseek_make(0);
  P_open(&pads, 0, io_disc);

  entry_t_init(pads, &rep);
  entry_t_pd_init(pads, &pd);
  entry_t_m_init(pads, &mask, P_CheckAndSet);

  if (P_ERR == P_io_fopen(pads, fname)) {
    error(2, "*** P_io_fopen failed ***");
    return -1;
  }

  entry_t_hist_init(pads, &h);
  default_hist.toFloat = 0;
  default_hist.fromFloat = 0;
  entry_t_hist_setPara(pads, &h, &default_hist);
  while (!P_io_at_eof(pads)) {
    entry_t_read(pads, &mask, &pd, &rep);
    entry_t_hist_add(pads, &h, &pd, &rep, &isFull);
    if (isFull != 0) entry_t_hist_reportFull(pads, "", 0, 0, &h);
  }
  entry_t_hist_reportAll(pads, "", 0, 0, &h);

  P_io_close(pads);
  entry_t_cleanup(pads, &rep);
  entry_t_pd_cleanup(pads, &pd);
  entry_t_hist_cleanup(pads, &h);
  P_close(pads);
  return 0;
}
```

Figure 17.3: Simple use of histogram functions for the `entry_t` type from CLF data.

`entry_t_hist_reportAll` Writes summary report for all histograms to screen.

Figure 17.3 illustrates a sample use of histogram functions for printing a summary of CLF `entry_t`s.

17.2 Customization

Users are allowed to customize various aspects of histogram by setting the appropriate field in the histogram data structure, which contains:

`INIT_N` is a `Puint64` denoting the number of values for histogram to summarize. If the number of values in the data source exceeds `INIT_N`, histograms will be built on each `INIT_N` data values respectively, until the end of data source is reached.

`INIT_B` is a `Puint32` denoting the number of buckets in final histogram. As `INIT_B` increases, accuracy of final histogram approximation is increased, while more time and space is consumed. The default value is 10.

`INIT_M` is a `Pint64` denoting an upper bound of data values in data source. Time consumed increases in poly-logrithm of `INIT_M`, so `INIT_M` can be set very large if little about data values in data source is known.

`INIT_ISE` is a `Pint8` denoting whether buckets in the final histogram are required to be of the same width. If `INIT_ISE` is set to be non-zero, all buckets have equal width. In this case, the time needed is in linear-`INIT_N`, and only constant space will be used. However, the result histogram will have less accuracy.

`INIT_ISO` is a `Pint8` denoting whether final histogram is required to be optimal or not. This parameter is valid only when `INIT_ISE` is zero. If `INIT_ISO` is set to be non-zero, the result histogram will be the most accurate one among all `INIT_B` bucket histograms. However, the time needed is in cubic-`INIT_N`, which could be extremely slow, and the space needed is in linear-`INIT_N`, since all data values in each `INIT_N` section are required to be stored.

`INIT_n` is a `Pint8` denoting whether norm 1 or norm 2 is used to measure accuracy of final histogram. Currently, norm 1 measurement is supported only when all the data values are stored. In other words, it is supported only when `INIT_ISE` is zero, and `INIT_ISO` is non-zero.

`INIT_e` is a `Pfloat64` denoting error tolerance of the final histogram. This parameter is valid only when non-optimal result is allowed, namely both `INIT_ISE` and `INIT_ISO` are zeroes. The final histogram will be guaranteed to be no worse than poly- (1+`INIT_e`) times of the optimal one, but the time and space needed increase as the error tolerance decreases.

`INIT_scale` is a `Pint64` denoting scale factor for each data value. This parameter is important for computing, but will not affect the final result. For example, if the data source can take values up to `64` bits, the overall `SSE` could need as many as `128` bits, which exceeds the representation limit of PADS. In this case, `INIT_scale` is needed.

`INIT_maxPortion` is a `Pint8` denoting the maximal number of stored histograms. If user forgets to report a finished histogram, or the inner histogram is finished while the outer one is not in nested cases, the finished histogram will go into a stored histogram list. If a new histogram is required to build, while the number of stored histograms is already `INIT_maxPortion`, the list will be cleared and a warning will be reported.

`entry_t_toFloat` is a function pointer, taking `entry_t` as input parameter, and returning corresponding `Pfloat64`. Histograms will handle `Pfloat64` type data value only. Any type with a well-defined conversion function to `Pfloat64` is considered as a meaningful type, and could be summarized correctly by histograms. By default, all base types other than `Pstring` in PADS have conversion functions to `Pfloat64`. Users are allowed to write their own conversion function for each field by defining macro `EXTRA_INIT_CODE`. If zero is assigned to this pointer, those default conversion functions will be used.

`entry_t_fromFloat` is a function pointer, taking `Pfloat64` as input parameter, and returning corresponding `entry_t` type. Any type without a well-defined conversion function from `Pfloat64` may not be printed correctly. By default, all base types other than `Pstring` in PADS have conversion functions from `Pfloat64`. Users are allowed to write their own conversion function for each field by defining macro `EXTRA_INIT_CODE`. If zero is assigned to this pointer, those default conversion functions will be used.

17.3 Template Program

Because generating a histogram report from a PADS description is a very routine task, PADS provides a template program to automate the task for common data formats. In particular, the template applies to data that can be viewed as an optional header followed by a sequence of records. Note that any data source that can be read entirely into memory fits this pattern by considering the source to have no header and a single body record.

When instantiated, the template program takes an optional command-line argument specifying the path to the data source. If no argument is given, it uses a default location for the data specified by the template user. The template first reads the optional header, then reads each record and inserts the value of each meanful field into histogram until either the data source is exhuasted or the end of a portion is reached, at which point it prints the histogram report to standard io. The following list describes the macros used by histogram template:

`DEF_INPUT_FILE` If defined, this macro specifies a string representation of the path to the default data source. If no path to the data is supplied at the command-line, this is the location used for input data.

`EXTRA_BEGIN_CODE` If defined, this macro points to a C statement that will be executed after all initialization code is performed, but before the optional header is read.

`EXTRA_DECLS` This optional macro defines additional C declarations that proceed all template code.

`EXTRA_DONE_CODE` If defined, this macro points to a C statement that will be executed after generating the accumulator report.

`EXTRA_INIT_CODE` This optional macro defines additional C codes that customize histogram data structure for different fields.

`EXTRA_READ_ARGS` If the type of the repeated record was parameterized, this macro allows the user to supply corresponding parameters.

`IO_DISC_MK` If defined, this macro specifies the interpretation of **`Precord`** by indicating which IO discipline the system should install. It specifies the discipline by naming the function to create the discipline. Section 15.2 describes the available IO discipline creation functions. If the user does not define this macro, the system installs the IO discipline corresponding to new-line terminated ASCII records.

`PADS_HDR_TY` Intuitively, this macro defines the type of the header record in the data source. This macro need only be defined if the data source has a header record. It defines a function used by the template program to generate the various function and type names derived from the name of the header record type, $i.e.$, the type of the associated in-memory representation, mask, parse descriptor, read function, etc.

`PADS_TY` Intuitively, this macro defines the type of the repeated record in the data source, $i.e.$, the type of the value to be summarized. This macro must be defined to use the histogram template. It defines a function used by the template program to generate the various function and type names derived from

the name of the record type, $i.e.$, the type of the associated in-memory representation, mask, parse descriptor, read function, etc.

`READ_MASK` This macro specifies the mask to use in reading the repeated record. If not defined by the user, the template uses the value `P_CheckAndSet`.

Chapter 18

Cluster

Clustering program divides data into several groups based on certain distribution. It summarizes the data by recording specified features of each group. Clustering is built for each meaningful piece of a PADS description. Figure 18.1 is an example report for a web server log data. In this particular run, maximal 3 clusterings are built for all the data values seen in the data source.

18.1 Operations

Figure 18.2 shows the clustering functions declared for a PADS type. These functions have the following behaviors:

`entry_t_hist_init` Initializes clustering data structure. This function must be called before any data can be added to the programme.

`entry_t_hist_setPara` Customizes clustering data structure. For the distribution function and two conversion functions (specified below), user needs to set the corresponding fields explicitly. This function must be called to make any customization effected.

`entry_t_hist_reset` Reinitializes clustering data structure. This function can be used to set any point of the data source as the start point of a new run. But it can't be used to reset any previous defined parameters.

`entry_t_hist_cleanup` Deallocates all memory associated with clustering.

`entry_t_hist_add` Inserts a data value. This function is called once a new record is coming. Any data type with an associated mapping function to `Pfloat64` is considered as a meaningful type. This function tracks fields with meaningful type and legal values only.

`entry_t_hist_report2io` Writes summary report for clustering `c` to `*outstr`.

`entry_t_hist_report` Writes summary report for clustering `c` to screen.

Figure 18.3 illustrates a sample use of clustering functions for printing a summary of CLF `entry_t`.

```
[Describing each tag arm of <top>.host]

================================================================================================
<top>.host.resolved : array nIP of Puint8
================================================================================================
Array lengths:
Clustering based distribution: User defined distribution.
mean 4, and variance 0, containing 4 elements.
================================================================================================
Possible anormality based on probability 0.010000:
Possible anormality based on clustering elements number 0.100000:

------------------------------------------------------------------------------------------------
allArrayElts : uint8
------------------------------------------------------------------------------------------------

Clustering based distribution: User defined distribution.
mean 128, and variance 77, containing 8 elements.
mean 136, and variance 0, containing 4 elements.
mean 97, and variance 0, containing 4 elements.
================================================================================================
Possible anormality based on probability 0.010000:
Data (around): 49
Data (around): 207
Data (around): 49
Data (around): 207
Data (around): 50
Data (around): 207
Data (around): 50
Possible anormality based on clustering elements number 0.100000:

================================================================================================
<top>.host.symbolic : array sIP of Pstring_SE
================================================================================================
Array lengths:
Clustering based distribution: User defined distribution.
mean 4, and variance 0, containing 7 elements.
================================================================================================
Possible anormality based on probability 0.010000:
Possible anormality based on clustering elements number 0.100000:

------------------------------------------------------------------------------------------------
allArrayElts : string
------------------------------------------------------------------------------------------------

Clustering based distribution: User defined distribution.
mean non defined., and variance non defined., containing 28 elements.
================================================================================================
Possible anormality based on probability 0.010000:
Possible anormality based on clustering elements number 0.100000:
. . . . . .
```

Figure 18.1: Portion of clustering report for web server log data.

```
Perror_t entry_t_cluster_init (P_t *pads,entry_t_cluster *h);
Perror_t entry_t_cluster_setPara (P_t *pads,entry_t_cluster *h,P_cluster *d_cluster);
Perror_t entry_t_cluster_reset (P_t *pads,entry_t_cluster *h);
Perror_t entry_t_cluster_cleanup (P_t *pads,entry_t_cluster *h);
Perror_t entry_t_cluster_add (P_t *pads,entry_t_cluster *h,Pbase_pd
*pd,entry_t *rep,Puint32 *isFull);
Perror_t entry_t_cluster_report2io (P_t *pads, Sfio_t *ourstr,
const char *prefix, const char *what, int nst, entry_t_cluster *h);
Perror_t entry_t_cluster_report (P_t *pads, const char *prefix,
const char *what, int nst, entry_t_cluster *h);
```

Figure 18.2: Clustering functions generated for the `entry_t` type.

18.2 Customization

Users are allowed to customize various aspects of clustering by setting the appropriate field in the clustering data structure, which contains:

`INIT_CTYPE` is an enumeration denoting the type of the underlying distribution for each clustering. Built-in distributions include K_mean, Gaussian distribution, Exponential distribution and Laplace distribution. Users are allowed to add any distributions, which could be fully characterized by mean and variance, by setting the field `Distri_fn`, which will be specified later.

`INIT_K` is a `Puint32` denoting the maximal number of clusterings users want to use to divide the date source. Together with `INIT_CTYPE`, it decides the underlying model of the data source.

`INIT_OPEN` is a `Pfloat64` denoting the probability threshold for opening a new clustering. A new clustering will be opened for a coming data value, if and only if, the number of current clusterings is less than `INIT_K` and the probabilities it falls in all current clusterings are less than `INIT_OPEN`.

`INIT_INITVAR` is a `Pfloat64` denoting the initial variance for each clustering. It takes effect only before the second data item is inserted. After that, the variance of each clustering will be fully decided by its elements.

`INIT_ANORM_POS` is a `Pfloat64` denoting the probability threshold for detecting anormality. A data value will be reported as anormality if no new clustering is opened for it, and the probabilities it falls in all existing clusterings are less than `INIT_ANORM_POS`. The data value detected later in the data source is expected to be more accurate than the one detected at the beginning of the data source.

`INIT_ANORM_NUM` is a `Pfloat64` denoting the element number threshold for detecting anormality. A whole clustering will be reported as anormality if the number of its elements is less than `INIT_ANORM_NUM` of the total number of data items in the data source.

`entry_t_probFn` is a function pointer, taking mean, variance and data value as input, and returning corresponding probability of that data value, according to mean and variance. Users could define their own distribution for each clustering, as long as the distribution is fully specified by mean and variance. Doing this, they need: first, set `INIT_CTYPE` to be `OTHERS`; then, use `EXTRA_INIT_CODE` to define their own distribution function, and assign them to this pointer. If `OTHERS` is set to `INIT_CTYPE`, and zero is set to this pointer, Gaussian distribution will be used.

`entry_t_toFloat` is a function pointer, taking `entry_t` as input parameter, and returning corresponding `Pfloat64`. Clusterings will handle `Pfloat64` type data value only. Any type with a well-defined conversion function to `Pfloat64` is considered as a meaningful type, and could be summarized correctly by clusterings. By default, all base types other than `Pstring` in PADS have conversion

```
#include "wsl.h"
#define DEF_INPUT_FILE  "data/wsl"

int main(int argc, char** argv) {
  P_t                    *pads;
  Pio_disc_t            *io_disc;
  P_cluster              default_cluster;
  entry_t                rep;
  entry_t_pd            pd;
  entry_t_m             mask;
  entry_t_cluster       c;
  Puint32                 isFull;
  char               *fname = DEF_INPUT_FILE;

  io_disc = P_nlrec_noseek_make(0);
  P_open(&pads, 0, io_disc);

  entry_t_init(pads, &rep);
  entry_t_pd_init(pads, &pd);
  entry_t_m_init(pads, &mask, P_CheckAndSet);

  if (P_ERR == P_io_fopen(pads, fname)) {
    error(2, "*** P_io_fopen failed ***");
    return -1;
  }

  entry_t_cluster_init(pads, &h);
  default_cluster.toFloat = 0;
  default_cluster.fromFloat = 0;
  default_cluster.Distir_fn = 0;
  entry_t_cluster_setPara(pads, &h, &default_cluster);
  while (!P_io_at_eof(pads)) {
    entry_t_read(pads, &mask, &pd, &rep);
    entry_t_cluster_add(pads, &h, &pd, &rep, isFull);
  };
  entry_t_cluster_report(pads, "", 0, 0, &h);

  P_io_close(pads);
  entry_t_cleanup(pads, &rep);
  entry_t_pd_cleanup(pads, &pd);
  entry_t_cluster_cleanup(pads, &h);
  P_close(pads);
  return 0;
}
```

Figure 18.3: Simple use of clustering functions for the `entry_t` type from CLF data.

functions to `Pfloat64`. Users are allowed to write their own conversion function for each field by defining macro `EXTRA_INIT_CODE`. If zero is assigned to this pointer, those default conversion functions will be used.

`entry_t_fromFloat` is a function pointer, taking `Pfloat64` as input parameter, and returning corresponding `entry_t` type. Any type without a well-defined conversion function from `Pfloat64` may not be printed correctly. By default, all base types other than `Pstring` in PADS have conversion functions from `Pfloat64`. Users are allowed to write their own conversion function for each field by defining macro `EXTRA_INIT_CODE`. If zero is assigned to this pointer, those default conversion functions will be used.

18.3 Template Program

Because generating a clustering report from a PADS description is a very routine task, PADS provides a template program to automate the task for common data formats. In particular, the template applies to data that can be viewed as an optional header followed by a sequence of records. Note that any data source that can be read entirely into memory fits this pattern by considering the source to have no header and a single body record.

When instantiated, the template program takes an optional command-line argument specifying the path to the data source. If no argument is given, it uses a default location for the data specified by the template user. The template first reads the optional header, then reads each record and inserts the value of each meanful field into clustering until either the data source is exhuasted or the end of a portion is reached, at which point it prints the clustering report to standard io. The following list describes the macros used by clustering template:

`DEF_INPUT_FILE` If defined, this macro specifies a string representation of the path to the default data source. If no path to the data is supplied at the command-line, this is the location used for input data.

`EXTRA_BEGIN_CODE` If defined, this macro points to a C statement that will be executed after all initialization code is performed, but before the optional header is read.

`EXTRA_DECLS` This optional macro defines additional C declarations that proceed all template code.

`EXTRA_DONE_CODE` If defined, this macro points to a C statement that will be executed after generating the accumulator report.

`EXTRA_INIT_CODE` This optional macro defines additional C codes that customize clustering data structure for different fields.

`EXTRA_READ_ARGS` If the type of the repeated record was parameterized, this macro allows the user to supply corresponding parameters.

`IO_DISC_MK` If defined, this macro specifies the interpretation of **`Precord`** by indicating which IO discpline the system should install. It specifies the discipline by naming the function to create the discipline. Section 15.2 describes the available IO discipline creation functions. If the user does not define this macro, the system installs the IO discipline corresponding to new-line terminated ASCII records.

`PADS_HDR_TY` Intuitively, this macro defines the type of the header record in the data source. This macro need only be defined if the data source has a header record. It defines a function used by the template program to generate the various function and type names derived from the name of the header record type, *i.e.*, the type of the associated in-memory representation, mask, parse descriptor, read function, etc.

`PADS_TY` Intuitively, this macro defines the type of the repeated record in the data source, $i.e.$, the type of the value to be summarized. This macro must be defined to use the clustering template. It defines a function used by the template program to generate the various function and type names derived from the name of the record type, $i.e.$, the type of the associated in-memory representation, mask, parse descriptor, read function, etc.

`READ_MASK` This macro specifies the mask to use in reading the repeated record. If not defined by the user, the template uses the value `P_CheckAndSet`.

Chapter 19

Formatting

Formatting functions provide support for pretty printing PADS values in delimiter separated forms suitable for loading into relational databases. For example, Figure 2.10 illustrates using generated formatting functions to convert the CLF data from Figure 2.1 into a pipe-delimited form.

19.1 Operations

```
ssize_t entry_t_fmt2buf_final (P_t *pads,Pbyte *buf,size_t buf_len,int *buf_full,
                               int *requestedOut,char const *delims,entry_t_m *m,
                               entry_t_pd *pd,entry_t *rep);

ssize_t entry_t_fmt2buf (P_t *pads,Pbyte *buf,size_t buf_len,int *buf_full,
                         int *requestedOut,char const *delims,
                         entry_t_m *m,entry_t_pd *pd,entry_t *rep);
ssize_t entry_t_fmt2io (P_t *pads,Sfio_t *io,int *requestedOut,
                        char const *delims,entry_t_m *m,entry_t_pd *pd,
                        entry_t *rep);
```

Figure 19.1: Formatting functions generated for the `entry_t` type.

Chapter 20

XML

This chapter is under development.

Chapter 21

Filters

Filters partition an input file into two output files: one with data conforming to the specification, the other with data containing errors. Filters apply to data formats that contain an optional header followed by a sequence of records.

21.1 Template Program

Because generating a filter from a PADS description is a very routine task, PADS provides a template program to automate the task for common data formats. In particular, the template applies to data that can be viewed as an optional header followed by a sequence of records.

When instantiated, the template program takes an optional command-line argument specifying the path to the data source. If no argument is given, it uses a default location for the data specified by the template user. The location for the clean and error records can be set in the template program.

The template first reads the optional header, echoing it to either the clean or the error file, depending upon the resulting parse descriptor. It then reads each record, echoing it to the approporiate file until the data source is exhausted.

Like the accumulator template, the filter template is a C header file parameterized by a number of macros that permit the user to customize the template by defining appropriate values for these macros. The following list describes the macros used by the filter template:

`DATE_IN_FMT` If defined, this macro sets the default input format for dates described by `Pdate`. See Section 15.1.12 for more information.

`DATE_OUT_FMT` If defined, this macro sets the default output format for `Pdate` and `Pdate_explicit`. See Section 15.1.13 for more information.

`DEF_INPUT_FILE` If defined, this macros specifies a string representation of the path to the default data source. If no path to the data is supplied at the command-line, this is the location used for input data.

`EXTRA_BAD_READ_CODE` If defined, this macro points to a C statement that will be executed after any body record containing an error.

`EXTRA_BEGIN_CODE` If defined, this macro points to a C statement that will be executed after all initialization code is performed, but before the optional header is read.

`EXTRA_DECLS` This optional macro defines additional C declarations that proceed all accumulator code.

EXTRA_DONE_CODE If defined, this macro points to a C statement that will be executed after generating the accumulator report.

EXTRA_GOOD_READ_CODE If defined, this macro points to a C statement that will be executed after any body record not containing an error.

EXTRA_HEADER_READ_ARGS If the type of the header record was parameterized, this macro allows the user to supply corresponding parameters.

EXTRA_READ_ARGS If the type of the repeated record was parameterized, this macro allows the user to supply corresponding parameters.

IN_TIME_ZONE If set, this macro specifies the input time zone of date types that do not include time zone information. See Section 15.1.10 for more detail.

IO_DISC_MK If defined, this macro specifies the interpretation of **`Precord`** by indicating which IO discpline the system should install. It specifies the discipline by naming the function to create the discipline. Section 15.2 describes the available IO discipline creation functions. If the user does not define this macro, the system installs the IO discipline corresponding to new-line terminated ASCII records.

MAX_RECS If defined, this macro specifies an integer that limits the number of repeated records that the accumulator program should read.

OUT_TIME_ZONE If set, this macro specifies the output time zone of date types. See Section 15.1.11 for more detail.

PADS_HDR_TY Intuitively, this macro defines the type of the header record in the data source. This macro need only be defined if the data source has a header record. It defines a function used by the template program to generate the various function and type names derived from the name of the header record type, *i.e.*, the type of the associated in-memory representation, mask, parse descriptor, read function, *etc.*

PADS_TY Intuitively, this macro defines the type of the repeated record in the data source, *i.e.*, the type of the value to be accumulated. This macro must be defined to use the accumulator template. It defines a function used by the template program to generate the various function and type names derived from the name of the record type, *i.e.*, the type of the associated in-memory representation, mask, parse descriptor, read function, *etc.*

READ_MASK This macro specifies the mask to use in reading the repeated record. If not defined by the user, the template uses the value `P_CheckAndSet`.

TIME_IN_FMT If defined, this macro sets the default input format for `Ptime`. See Section 15.1.12 for more information.

TIME_OUT_FMT If defined, this macro sets the default output format for `Ptime` and `Ptime_explicit`. See Section 15.1.13 for more information.

TIMESTAMP_IN_FMT If defined, this macro sets the default input format for `Ptimestamp`. See Section 15.1.12 for more information.

TIMESTAMP_OUT_FMT If defined, this macro sets the default output format for the PADS types `Ptimestamp` and `Ptimestamp_explicit`. See Section 15.1.13 for more information.

WSPACE_OK If defined, this macro indicates that leading white space for variable-width ASCII integers is okay, as well as leading and trailing white space for fixed-width ASCII integers.

Bibliography

[asd] Abstract syntax description language. `http://sourceforge.net/projects/asdl`.

[Bac02] Back, G. DataScript - A specification and scripting language for binary data. In *Proceedings of Generative Programming and Component Engineering*, vol. 2487. LNCS, 2002, pp. 66–77.

[CFP+04] Cortes, C., K. Fisher, D. Pregibon, A. Rogers, and F. Smith. Hancock: A language for analyzing transactional data streams. *ACM Trans. Program. Lang. Syst.*, **26**(2), 2004, pp. 301–338.

[CGJ+02] Cranor, C., Y. Gao, T. Johnson, V. Shkapenyuk, and O. Spatscheck. Gigascope: High performance network monitoring with an SQL interface. In *SIGMOD*. ACM, 2002.

[CP98] Cortes, C. and D. Pregibon. Giga mining. In *KDD*, 1998.

[CP99] Cortes, C. and D. Pregibon. Information mining platform: An infrastructure for KDD rapid deployment. In *KDD*, 1999.

[Dub01] Dubuisson, O. *ASN.1: Communication between heterogeneous systems*. Morgan Kaufmann, 2001.

[erla] Erlang bit syntax. `http://www.erlang.se/euc/99/binaries.ps`.

[erlb] Proposals for and experiments with an Erlang bit syntax. http://lambda.weblogs.com/discuss/msgReader$5185.

[KR01] Krishnamurthy, B. and J. Rexford. *Web Protocols and Practice*. Addison Wesley, 2001.

[KW00] Krishnamurthy, B. and J. Wang. On network-aware clustering of web clients. In *Proceedings of SIGCOMM 2000*. ACM, 2000.

[KW02] Krishnamurthy, B. and C. Wills. Improving web experience by client characterization driven server adaptation. In *Proceedings of WWW 2002*. ACM, 2002.

[MC98] McCann, P. and S. Chandra. PacketTypes: Abstract specification of network protocol messages. In *ACM Conference of Special Interest Group on Data Communications (SIGCOMM)*, August 1998, pp. 321–333.

[net] Cisco netflow. `http://www.cisco.com/warp/public/732/Tech/nmp/netflow/index.shtml`.

[New] Tree formats. Workshop on molecular evolution. `http://workshop.molecularevolution.org/resources/fileformats/tree_formats.php`.

Appendix A

Error codes

The values of type `Perror_t` are:

Name	Value	Meaning
`P_OK`	`0`	No error
`P_ERR`	`-1`	Some error

The values of type `PerrCode_t` are listed below. The "Has location" field indicates whether the parse descriptor contains a corresponding location in the source file.

Name	Has location	Meaning
`P_NOT_PARSED`	No	Data not yet parsed
`P_NO_ERR`	No	Value read, no error detected
`P_SKIPPED`	No	
`P_UNEXPECTED_ERR`	No	
`P_BAD_PARAM`	No	
`P_SYS_ERR`	No	
`P_IO_ERR`	No	
`P_CHKPOINT_ERR`	No	
`P_COMMIT_ERR`	No	
`P_RESTORE_ERR`	No	
`P_ALLOC_ERR`	No	
`P_FORWARD_ERR`	No	
`P_PANIC_SKIPPED`	No	Value skipped because of panic
`P_FMT_EMPTY_DELIM_ERR`	No	
`P_INVALID_FUNCTION_CALL`	No	
`P_SMART_NODE_ERR`	No	General error relating to smart nodes
`P_FAILWITH_ERR`	No	Error requiring an ocaml exception

Name	Has location	Meaning
P_USER_CONSTRAINT_VIOLATION	Yes	
P_MISSING_LITERAL	Yes	
P_ARRAY_ELEM_ERR	Yes	
P_ARRAY_SEP_ERR	Yes	
P_ARRAY_TERM_ERR	Yes	
P_ARRAY_SIZE_ERR	Yes	
P_ARRAY_SEP_TERM_SAME_ERR	Yes	
P_ARRAY_USER_CONSTRAINT_ERR	Yes	
P_ARRAY_MIN_BIGGER_THAN_MAX_ERR	Yes	
P_ARRAY_MIN_NEGATIVE	Yes	
P_ARRAY_MAX_NEGATIVE	Yes	
P_ARRAY_EXTRA_BEFORE_SEP	Yes	
P_ARRAY_EXTRA_BEFORE_TERM	Yes	
P_STRUCT_FIELD_ERR	Yes	
P_STRUCT_EXTRA_BEFORE_SEP	Yes	
P_UNION_MATCH_ERR	Yes	
P_OPTION_MATCH_ERR	Yes	
P_ENUM_MATCH_ERR	Yes	
P_TYPEDEF_CONSTRAINT_ERR	Yes	
P_AT_EOF	Yes	
P_AT_EOR	Yes	
P_EXTRA_BEFORE_EOR	Yes	
P_EOF_BEFORE_EOR	Yes	
P_COUNT_MAX_LIMIT	Yes	
P_RANGE	Yes	
P_INVALID_A_NUM	Yes	Invalid ASCII number
P_INVALID_E_NUM	Yes	Invalid EBCDIC number
P_INVALID_EBC_NUM	Yes	Invalid EBC number
P_INVALID_BCD_NUM	Yes	Invalid BCD number
P_INVALID_CHARSET	Yes	
P_INVALID_WIDTH	Yes	
P_CHAR_LIT_NOT_FOUND	Yes	
P_STR_LIT_NOT_FOUND	Yes	
P_REGEXP_NOT_FOUND	Yes	
P_INVALID_REGEXP	Yes	
P_WIDTH_NOT_AVAILABLE	Yes	
P_INVALID_TIMESTAMP	Yes	
P_INVALID_DATE	Yes	
P_INVALID_TIME	Yes	
P_INVALID_IP	Yes	
P_INVALID_IP_RANGE	Yes	

Appendix B

All PADS Base Types

This appendix gives the full set of signatures for the read, write, format, and accumulation functions for each base type that is part of the standard PADS distribution. It also describes how read behavior differs with different setting for the read mask, as well as error behavior for each possible error that can occur during reading.

This information is also available in the type-specific include files which can be found under the `padsc/include/ptypes` directory.

B.1 Counting: Character encodings

```
/* ================================================================================
 * CHAR COUNTING FUNCTIONS
 *
 * DEFAULT                     ASCII                       EBCDIC
 * --------------------------  --------------------------  ------------------------
 * PcountX_read                Pa_countX_read              Pe_countX_read
 * PcountXtoY                  Pa_countXtoY_read           Pe_countXtoY_read
 *
 * countX counts occurrences of char x between the current IO cursor
 * and the first EOR or EOF, while countXtoY counts occurrences of x
 * between the current IO cursor and the first occurrence of char y.
 * x and y are always specified as ASCII chars.  They are converted to
 * EBCDIC if the EBCDIC form is used or if the DEFAULT form is used
 * and pads->disc->def->charset is Pcharset_EBCDIC.
 *
 * If parameter count_max is non-zero, then the count functions also
 * stop counting after scanning count_max characters, in which case an
 * error is returned.  If the IO discipline is not record-based and
 * count_max is zero, an error is returned immediately: you *must*
 * specify a count_max > 0 when using an IO discipline that has no
 * records.
 */
```

```
/*
 * For countX, if param eor_required is non-zero, then encountering EOF
 * before EOR produces an error.
 *
 * These functions do not change the IO cursor position.
 *
 * countX outcomes:
 *   1. IO cursor is already at EOF and eor_required is non-zero:
 *     + pd->loc.b/e set to EOF 'location'
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_AT_EOF,
 *         pd->nerr set to 1, and an error is reported
 *     + P_ERR returned
 *   2. EOF is encountered before EOR and eor_required is non-zero:
 *     + pd->loc.b/e set to current IO cursor location
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_EOF_BEFORE_EOR,
 *         pd->nerr set to 1, and an error is reported
 *     + P_ERR returned
 *   3. count_max is > 0 and count_max limit is reached before x
 *       or EOR or EOF:
 *     + pd->loc.b/e set to current IO cursor location
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_COUNT_MAX_LIMIT,
 *         pd->nerr set to 1, and an error is reported
 *     + P_ERR returned
 *   4. EOR is encountered, or EOF is encounterd and eor_required is zero:
 *     + (*res_out) is set to the number of occurrences of x from the
 *        IO cursor to EOR/EOF
 *     + P_OK returned
 */



/*
 * countXtoY outcomes:
 *   1. IO cursor is already at EOF
 *     + pd->loc.b/e set to EOF 'location'
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_AT_EOF,
 *         pd->nerr set to 1, and an error is reported
 *     + P_ERR returned
 *   2. y is not found before EOR or EOF is hit
 *     + pd->loc.b/e set to current IO cursor location
 *     + if P_Test_NotIgnore(*m), pd->errCode set to
 *         P_CHAR_LIT_NOT_FOUND, pd->nerr set to 1,
 *         and an error is reported
 *     + P_ERR returned
 *   3. y is not found and count_max > 0 and count_max limit is hit
 *     + pd->loc.b/e set to current IO cursor location
 *     + if P_Test_NotIgnore(*), pd->errCode set to P_COUNT_MAX_LIMIT,
 *         pd->nerr set to 1, and an error is reported
 *     P_ERR returned
 *   4. Char y is found
 *     + (*res_out) is set to the number of occurrences of x
 *       from the IO cursor to first y
 *     + P_OK returned
 */



Perror_t Pa_countX_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint8 x, int eor_required, size_t count_max
                    );

Perror_t Pa_countXtoY_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint8 x, Puint8 y, size_t count_max
                    );
```

```
Perror_t Pe_countX_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint8 x, int eor_required, size_t count_max
                    );

Perror_t Pe_countXtoY_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint8 x, Puint8 y, size_t count_max
                    );
```

```
Perror_t PcountX_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint8 x, int eor_required, size_t count_max
                    );

Perror_t PcountXtoY_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint8 x, Puint8 y, size_t count_max
                    );
```

```
ssize_t PcountX_write2io(P_t *pads, Sfio_t *io, Puint8 x, int eor_required,
                    size_t count_max, Pbase_pd *pd, Pint32 *val
                    );

ssize_t PcountX_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Puint8 x, int eor_required, size_t count_max, Pbase_pd *pd,
                    Pint32 *val
                    );

ssize_t PcountXtoY_write2io(P_t *pads, Sfio_t *io, Puint8 x, Puint8 y,
                    size_t count_max, Pbase_pd *pd, Pint32 *val
                    );

ssize_t PcountXtoY_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Puint8 x, Puint8 y, size_t count_max, Pbase_pd *pd, Pint32 *val
                    );
```

```
ssize_t PcountX_write_xml_2io(P_t *pads, Sfio_t *io, Puint8 x, int eor_required,
                    size_t count_max, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent
                    );

ssize_t PcountX_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Puint8 x, int eor_required, size_t count_max, Pbase_pd *pd,
                    Pint32 *val, const char *tag, int indent
                    );

ssize_t PcountXtoY_write_xml_2io(P_t *pads, Sfio_t *io, Puint8 x, Puint8 y,
                    size_t count_max, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent
                    );

ssize_t PcountXtoY_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Puint8 x, Puint8 y, size_t count_max, Pbase_pd *pd,
                    Pint32 *val, const char *tag, int indent
                    );
```

B.2 Chars: Character encodings

```
/* ================================================================================
 * READ
 */

/* ================================================================================
 * CHAR READ FUNCTIONS
 *
 * DEFAULT                  ASCII                    EBCDIC
 * -----------------------  -----------------------  ---------------------------
 * Pchar_read               Pa_char_read             Pe_char_read
 *
 * Cases:
 *  (1) A character is not available:
 *        + pd->loc set to the current IO position
 *        + if P_Test_NotIgnore(*m), pd->errCode is set to
 *            P_WIDTH_NOT_AVAILABLE, pd->nerr is set to 1,
 *            and an error is reported
 *        + the IO cursor is not advanced
 *        + P_ERR is returned
 *  (2) A character is available and P_Test_NotSet(*m)
 *        + the IO cursor is advanced by 1 byte
 *        + P_OK is returned
 *  (3) A character is available and P_Test_Set(*m)
 *        + (*c_out) is set to the ASCII equivalent of the character, where
 *             a conversion fom EBCDIC to ASCII occurs if the EBCDIC form is
 *             used or if the DEFAULT form is used and
 *             pads->disc->def_charset is Pcharset_EBCDIC.
 *        + the IO cursor is advanced by 1 byte
 *        + P_OK is returned
 */



Perror_t Pa_char_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pchar *c_out);




Perror_t Pe_char_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pchar *c_out);




Perror_t Pchar_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pchar *c_out);



/* ================================================================================
 * WRITE
 */

/* ================================================================================
 * CHAR WRITE FUNCTIONS
 * DEFAULT                    ASCII                      EBCDIC
 * -------------------------  -------------------------  ----------------------
 * Pchar_write2io             Pa_char_write2io           Pe_char_write2io
 *
 * Pchar_write2buf            Pa_char_write2buf          Pe_char_write2buf
 */
```

```
ssize_t Pa_char_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pchar *c);

ssize_t Pa_char_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pchar *c
                  );

ssize_t Pa_char_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pchar *c,
                  const char *tag, int indent
                  );

ssize_t Pa_char_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pchar *c, const char *tag,
                  int indent
                  );
```

```
ssize_t Pe_char_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pchar *c);

ssize_t Pe_char_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pchar *c
                  );

ssize_t Pe_char_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pchar *c,
                  const char *tag, int indent
                  );

ssize_t Pe_char_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pchar *c, const char *tag, int indent
                  );
```

```
ssize_t Pchar_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pchar *c);

ssize_t Pchar_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pchar *c
                  );

ssize_t Pchar_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pchar *c,
                  const char *tag, int indent
                  );

ssize_t Pchar_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pchar *c, const char *tag, int indent
                  );
```

```
ssize_t Pa_char_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pchar *rep
                    );

ssize_t Pa_char_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pchar *rep
                    );

ssize_t Pa_char_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Pchar *rep
                    );
```

```
ssize_t Pe_char_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pchar *rep
                    );

ssize_t Pe_char_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pchar *rep
                    );

ssize_t Pe_char_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Pchar *rep
                    );
```

```
ssize_t Pchar_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pchar *rep
                    );

ssize_t Pchar_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pchar *rep
                    );

ssize_t Pchar_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Pchar *rep
                    );
```

B.3 Strings: Character encodings

```
/* ============================================================================
 * READ
 */

/* ============================================================================
 * STRING READ FUNCTIONS
 *
 * DEFAULT                      ASCII                          EBCDIC
 * --------------------------  -----------------------------  -------------------
 * Pstring_FW_read             Pa_string_FW_read              Pe_string_FW_read
 * Pstring_read                Pa_string_read                 Pe_string_read
 * Pstring_ME_read             Pa_string_ME_read              Pe_string_ME_read
 * Pstring_CME_read            Pa_string_CME_read             Pe_string_CME_read
 * Pstring_SE_read             Pa_string_SE_read              Pe_string_SE_read
 * Pstring_CSE_read            Pa_string_CSE_read             Pe_string_CSE_read
 *
 * The string read functions each has a different way of specifying
 * the extent of the string:
 *   + all string_FW_read functions specify a fixed width.
 *     N.B.: width zero is allowed: the result is an empty string
 *       (and the IO cursor does not move)
 *   + all string_read functions specify a single stop character.
 *       if 0 (NULL) is used, then this will match a NULL in the data,
 *       and eor/eof will ALSO successfully terminate the string
 *   + all string_ME_read and string_CME_read functions specify a
 *       Match Expression (string includes all chars that match)
 *   + all string_SE_read and string_CSE_read specify a Stop Expression
 *       (string terminated by encountering 'stop chars' that match)
 */




/*
 * The ME/SE functions take a string containing a regular expression,
 * while the CME/CSE functions take a compiled form of regular
 * expression (see Pregexp_compile).
 *
 * stop chars and regular expressions are specified using ASCII, but
 * reading/matching occurs using converted EBCDIC forms if an EBCDIC
 * form is used or if a DEFAULT form is used and
 * pads->disc->def_charset is Pcharset_EBCDIC.
 *
 * For all stop cases, the stop char/chars are not included in the
 * resulting string.  Note that if the IO cursor is already at a stop
 * condition, then a string of length zero results.
 *
 * If an expected stop char/pattern/width is found, P_OK is returned.
 * If P_TestSet(*m) then:

 *   + (*s_out) is set to contain an in-memory string.  If the
 *     original data is ASCII, then s_out will either share the string
 *     or contain a copy of the string, depending on
 *     pads->disc->copy_strings.  If the original data is EBCDIC, s_out
 *     always contains a copy of the string that has been converted to
 *     ASCII.  N.B. : (*s_out) should have been initialized at some
 *     point prior using Pstring_init or one of the initializing
 *     P_STRING macros.  (It can be initialized once and re-used in
 *     string read calls many times.)
 */
```

```
/* Cleanup note: If pads->disc->copy_strings is non-zero, the memory
 * allocated in (*s_out) should ultimately be freed using
 * Pstring_cleanup.
 *
 * If an expected stop condition is not encountered, the
 * IO cursor position is unchanged.  Error codes used:
 *     P_WIDTH_NOT_AVAILABLE
 *     P_STOPCHAR_NOT_FOUND
 *     P_STOPREGEXP_NOT_FOUND
 *     P_INVALID_REGEXP
 *
 * EBCDIC Example: passing '|' (vertical bar, which is code 124 in
 * ASCII) to Pe_string_read as the stop char will result in a search
 * for the EBCDIC encoding of vertical bar (code 79 in EBCDIC), and
 * (*s_out) will be a string containing all chars between the IO
 * cursor and the EBCDIC vertical bar, with each EBCDIC char converted
 * to ASCII.
 */
```

```
Perror_t Pa_string_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                    size_t width
                    );

Perror_t Pa_string_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                    Pchar stopChar
                    );

Perror_t Pa_string_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                    const char *matchRegexp
                    );

Perror_t Pa_string_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                    Pregexp_t *matchRegexp
                    );

Perror_t Pa_string_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                    const char *stopRegexp
                    );

Perror_t Pa_string_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                    Pregexp_t *stopRegexp
                    );
```

```
Perror_t Pe_string_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   size_t width
                   );

Perror_t Pe_string_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   Pchar stopChar
                   );

Perror_t Pe_string_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   const char *matchRegexp
                   );

Perror_t Pe_string_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   Pregexp_t *matchRegexp
                   );

Perror_t Pe_string_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   const char *stopRegexp
                   );

Perror_t Pe_string_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   Pregexp_t *stopRegexp
                   );
```

```
Perror_t Pstring_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   size_t width
                   );

Perror_t Pstring_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   Pchar stopChar
                   );

Perror_t Pstring_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   const char *matchRegexp
                   );

Perror_t Pstring_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   Pregexp_t *matchRegexp
                   );

Perror_t Pstring_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   const char *stopRegexp
                   );

Perror_t Pstring_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pstring *s_out,
                   Pregexp_t *stopRegexp
                   );
```

```
/* ========================================================================
 * WRITE
 */

/* ========================================================================
 * STRING WRITE FUNCTIONS
 * DEFAULT                      ASCII                          EBCDIC
 * -------------------------  -----------------------------  ----------------------
 * Pstring_FW_write2io        Pa_string_FW_write2io          Pe_string_FW_write2io
 * Pstring_write2io           Pa_string_write2io             Pe_string_write2io
 * Pstring_ME_write2io        Pa_string_ME_write2io          Pe_string_ME_write2io
 * Pstring_CME_write2io       Pa_string_CME_write2io         Pe_string_CME_write2io
 * Pstring_SE_write2io        Pa_string_SE_write2io          Pe_string_SE_write2io
 * Pstring_CSE_write2io       Pa_string_CSE_write2io         Pe_string_CSE_write2io
 *
 * Pstring_FW_write2buf       Pa_string_FW_write2buf         Pe_string_FW_write2buf
 * Pstring_write2buf          Pa_string_write2buf            Pe_string_write2buf
 * Pstring_ME_write2buf       Pa_string_ME_write2buf         Pe_string_ME_write2buf
 * Pstring_CME_write2buf      Pa_string_CME_write2buf        Pe_string_CME_write2buf
 * Pstring_SE_write2buf       Pa_string_SE_write2buf         Pe_string_SE_write2buf
 * Pstring_CSE_write2buf      Pa_string_CSE_write2buf        Pe_string_CSE_write2buf
 */
```

```
ssize_t Pa_string_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    size_t width
                    );

ssize_t Pa_string_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, size_t width
                    );

ssize_t Pa_string_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    Pchar stopChar
                    );

ssize_t Pa_string_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, Pchar stopChar
                    );

ssize_t Pa_string_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *matchRegexp
                    );

ssize_t Pa_string_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, const char *matchRegexp
                    );
```

```
ssize_t Pa_string_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   Pregexp_t *matchRegexp
                   );

ssize_t Pa_string_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, Pregexp_t *matchRegexp
                   );

ssize_t Pa_string_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *stopRegexp
                   );

ssize_t Pa_string_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, const char *stopRegexp
                   );

ssize_t Pa_string_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   Pregexp_t *stopRegexp
                   );

ssize_t Pa_string_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, Pregexp_t *stopRegexp
                   );
```

```
ssize_t Pa_string_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *tag, int indent, size_t width
                   );

ssize_t Pa_string_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                   int indent, size_t width
                   );

ssize_t Pa_string_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *tag, int indent, Pchar stopChar
                   );

ssize_t Pa_string_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, const char *tag, int indent,
                   Pchar stopChar
                   );

ssize_t Pa_string_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *tag, int indent, const char *matchRegexp
                   );

ssize_t Pa_string_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                   int indent, const char *matchRegexp
                   );
```

```
ssize_t Pa_string_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pa_string_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pa_string_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, const char *stopRegexp
                    );

ssize_t Pa_string_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, const char *stopRegexp
                    );

ssize_t Pa_string_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, Pregexp_t *stopRegexp
                    );

ssize_t Pa_string_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_string_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    size_t width
                    );

ssize_t Pe_string_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, size_t width
                    );

ssize_t Pe_string_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    Pchar stopChar
                    );

ssize_t Pe_string_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, Pchar stopChar
                    );

ssize_t Pe_string_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *matchRegexp
                    );

ssize_t Pe_string_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, const char *matchRegexp
                    );
```

```
ssize_t Pe_string_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pe_string_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, Pregexp_t *matchRegexp
                    );

ssize_t Pe_string_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *stopRegexp
                    );

ssize_t Pe_string_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, const char *stopRegexp
                    );

ssize_t Pe_string_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    Pregexp_t *stopRegexp
                    );

ssize_t Pe_string_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_string_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_string_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pe_string_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, Pchar stopChar
                    );

ssize_t Pe_string_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, const char *tag, int indent,
                    Pchar stopChar
                    );

ssize_t Pe_string_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, const char *matchRegexp
                    );

ssize_t Pe_string_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, const char *matchRegexp
                    );
```

```
ssize_t Pe_string_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pe_string_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pe_string_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, const char *stopRegexp
                    );

ssize_t Pe_string_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, const char *stopRegexp
                    );

ssize_t Pe_string_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *tag, int indent, Pregexp_t *stopRegexp
                    );

ssize_t Pe_string_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                    int indent, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pstring_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    size_t width
                    );

ssize_t Pstring_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, size_t width
                    );

ssize_t Pstring_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    Pchar stopChar
                    );

ssize_t Pstring_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, Pchar stopChar
                    );

ssize_t Pstring_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                    const char *matchRegexp
                    );

ssize_t Pstring_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pstring *s, const char *matchRegexp
                    );
```

```
ssize_t Pstring_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   Pregexp_t *matchRegexp
                   );

ssize_t Pstring_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, Pregexp_t *matchRegexp
                   );

ssize_t Pstring_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *stopRegexp
                   );

ssize_t Pstring_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, const char *stopRegexp
                   );

ssize_t Pstring_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   Pregexp_t *stopRegexp
                   );

ssize_t Pstring_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, Pregexp_t *stopRegexp
                   );
```

```
ssize_t Pstring_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *tag, int indent, size_t width
                   );

ssize_t Pstring_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                   int indent, size_t width
                   );

ssize_t Pstring_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *tag, int indent, Pchar stopChar
                   );

ssize_t Pstring_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pstring *s, const char *tag, int indent,
                   Pchar stopChar
                   );

ssize_t Pstring_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                   const char *tag, int indent, const char *matchRegexp
                   );

ssize_t Pstring_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                   int indent, const char *matchRegexp
                   );
```

```
ssize_t Pstring_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                  const char *tag, int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pstring_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                  int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pstring_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                  const char *tag, int indent, const char *stopRegexp
                  );

ssize_t Pstring_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                  int indent, const char *stopRegexp
                  );

ssize_t Pstring_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pstring *s,
                  const char *tag, int indent, Pregexp_t *stopRegexp
                  );

ssize_t Pstring_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pstring *s, const char *tag,
                  int indent, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_string_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, size_t width
                  );

ssize_t Pa_string_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, size_t width
                  );

ssize_t Pa_string_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pchar stopChar
                  );

ssize_t Pa_string_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pchar stopChar
                  );

ssize_t Pa_string_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, const char *matchRegexp
                  );

ssize_t Pa_string_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, const char *matchRegexp
                  );
```

```
ssize_t Pa_string_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pa_string_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pa_string_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, const char *stopRegexp
                  );

ssize_t Pa_string_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, const char *stopRegexp
                  );

ssize_t Pa_string_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pa_string_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_string_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                  size_t width
                  );

ssize_t Pa_string_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                  Pchar stopChar
                  );

ssize_t Pa_string_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                  const char *matchRegexp
                  );

ssize_t Pa_string_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pa_string_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                  const char *stopRegexp
                  );

ssize_t Pa_string_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                  Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_string_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, size_t width
                  );

ssize_t Pe_string_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, size_t width
                  );

ssize_t Pe_string_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pchar stopChar
                  );

ssize_t Pe_string_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pchar stopChar
                  );

ssize_t Pe_string_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, const char *matchRegexp
                  );

ssize_t Pe_string_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, const char *matchRegexp
                  );
```

```
ssize_t Pe_string_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pe_string_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pe_string_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, const char *stopRegexp
                  );

ssize_t Pe_string_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, const char *stopRegexp
                  );

ssize_t Pe_string_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pstring *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pe_string_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Pstring *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_string_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    size_t width
                    );

ssize_t Pe_string_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    Pchar stopChar
                    );

ssize_t Pe_string_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    const char *matchRegexp
                    );

ssize_t Pe_string_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pe_string_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    const char *stopRegexp
                    );

ssize_t Pe_string_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pstring_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, size_t width
                    );

ssize_t Pstring_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, size_t width
                    );

ssize_t Pstring_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, Pchar stopChar
                    );

ssize_t Pstring_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, Pchar stopChar
                    );

ssize_t Pstring_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, const char *matchRegexp
                    );

ssize_t Pstring_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, const char *matchRegexp
                    );
```

```
ssize_t Pstring_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, Pregexp_t *matchRegexp
                    );

ssize_t Pstring_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Pstring *rep, Pregexp_t *matchRegexp
                    );

ssize_t Pstring_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, const char *stopRegexp
                    );

ssize_t Pstring_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, const char *stopRegexp
                    );

ssize_t Pstring_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pstring *rep, Pregexp_t *stopRegexp
                    );

ssize_t Pstring_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Pstring *rep, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pstring_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    size_t width
                    );

ssize_t Pstring_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Pstring *rep, Pchar stopChar
                    );

ssize_t Pstring_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    const char *matchRegexp
                    );

ssize_t Pstring_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pstring_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    const char *stopRegexp
                    );

ssize_t Pstring_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pstring *rep,
                    Pregexp_t *stopRegexp
                    );
```

B.4 IP addresses: Character encodings

```
/* ============================================================================
 * IP ADDRESS READ FUNCTIONS
 *
 * DEFAULT                        ASCII                          EBCDIC
 * -----------------------------  -----------------------------  -----------------------------
 * Pip_read                       Pa_ip_read                     Pe_ip_read
 *
 * Attempts to read a numeric IP address string, i.e., an IP address
 * form consisting of four numeric parts with values 0-255,
 * separated by ".", with an optional trailing dot.
 */



/* The result is a single Puint32 value with each part encoded in one
 * of the four bytes.  part1 is stored in the high-order byte, part4
 * in the low-order byte.  You can obtain each part using the macro
 *
 *   P_IP_PART(addr, part)
 *
 * where part must be from 1 to 4.
 *
 * The digit chars and "." char are read as EBCDIC chars if the EBCDIC
 * form is used or if the DEFAULT form is used and
 * pads->disc->def_charset is Pcharset_EBCDIC.  Otherwise the data is
 * read as ASCII chars.
 */



/* If the current IO cursor position points to a valid IP address string:
 *   + Sets (*res_out) to the resulting Puint32
 *   + advances the IO cursor position to just after the last legal
 *      character in the IP address string
 *   + returns P_OK
 * Otherwise:
 *   + pd->loc.b/e set to the IO cursor position
 *   + IO cursor is not advanced
 *   + if P_Test_NotIgnore(*m), pd->errCode set to P_INVALID_IP,
 *        pd->nerr set to 1, and an error is reported
 *   + returns P_ERR
 */



Perror_t Pa_ip_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);



Perror_t Pe_ip_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);



Perror_t Pip_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);
```

```
/* The helper macro P_IP_PART(addr, part) takes a Puint32 addr
 * and a part number from 1 to 4, and produces the specified part.
 */


/* ============================================================================
 * IP WRITE FUNCTIONS
 * DEFAULT                        ASCII                          EBCDIC
 * ------------------------------ ------------------------------ ------------------------------
 * Pip_write2io                   Pa_ip_write2io                 Pe_ip_write2io
 *
 * Pip_write2buf                  Pa_ip_write2buf                Pe_ip_write2buf
 */



ssize_t Pa_ip_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d);

ssize_t Pa_ip_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d
                    );

ssize_t Pa_ip_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent
                    );

ssize_t Pa_ip_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent
                    );




ssize_t Pe_ip_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d);

ssize_t Pe_ip_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d
                    );

ssize_t Pe_ip_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent
                    );

ssize_t Pe_ip_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent
                    );




ssize_t Pip_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d);

ssize_t Pip_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d
                    );

ssize_t Pip_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent
                    );

ssize_t Pip_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent
                    );
```

```
ssize_t Pa_ip_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep
                    );

ssize_t Pa_ip_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep
                    );

ssize_t Pa_ip_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep
                    );
```

```
ssize_t Pe_ip_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep
                    );

ssize_t Pe_ip_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep
                    );

ssize_t Pe_ip_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep
                    );
```

```
ssize_t Pip_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep);

ssize_t Pip_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep
                    );

ssize_t Pip_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep);
```

B.5 Timestamp, date, and time types: Character encodings

```
/* ============================================================================
 * READ FUNCTIONS
 *
 * DEFAULT                       ASCII                          EBCDIC
 * ----------------------------- ------------------------------ ------------------------------
 * Ptimestamp_explicit_FW_read   Pa_timestamp_explicit_FW_read   Pe_timestamp_explicit_FW_read
 * Ptimestamp_explicit_read      Pa_timestamp_explicit_read      Pe_timestamp_explicit_read
 * Ptimestamp_explicit_ME_read   Pa_timestamp_explicit_ME_read   Pe_timestamp_explicit_ME_read
 * Ptimestamp_explicit_CME_read  Pa_timestamp_explicit_CME_read  Pe_timestamp_explicit_CME_read
 * Ptimestamp_explicit_SE_read   Pa_timestamp_explicit_SE_read   Pe_timestamp_explicit_SE_read
 * Ptimestamp_explicit_CSE_read  Pa_timestamp_explicit_CSE_read  Pe_timestamp_explicit_CSE_read
 *
 * Pdate_explicit_FW_read        Pa_date_explicit_FW_read        Pe_date_explicit_FW_read
 * Pdate_explicit_read           Pa_date_explicit_read           Pe_date_explicit_read
 * Pdate_explicit_ME_read        Pa_date_explicit_ME_read        Pe_date_explicit_ME_read
 * Pdate_explicit_CME_read       Pa_date_explicit_CME_read       Pe_date_explicit_CME_read
 * Pdate_explicit_SE_read        Pa_date_explicit_SE_read        Pe_date_explicit_SE_read
 * Pdate_explicit_CSE_read       Pa_date_explicit_CSE_read       Pe_date_explicit_CSE_read
 *
 * Ptime_explicit_FW_read        Pa_time_explicit_FW_read        Pe_time_explicit_FW_read
 * Ptime_explicit_read           Pa_time_explicit_read           Pe_time_explicit_read
 * Ptime_explicit_ME_read        Pa_time_explicit_ME_read        Pe_time_explicit_ME_read
 * Ptime_explicit_CME_read       Pa_time_explicit_CME_read       Pe_time_explicit_CME_read
 * Ptime_explicit_SE_read        Pa_time_explicit_SE_read        Pe_time_explicit_SE_read
 * Ptime_explicit_CSE_read       Pa_time_explicit_CSE_read       Pe_time_explicit_CSE_read
 */
```

```
/*
 * Ptimestamp_FW_read            Pa_timestamp_FW_read            Pe_timestamp_FW_read
 * Ptimestamp_read               Pa_timestamp_read               Pe_timestamp_read
 * Ptimestamp_ME_read            Pa_timestamp_ME_read            Pe_timestamp_ME_read
 * Ptimestamp_CME_read           Pa_timestamp_CME_read           Pe_timestamp_CME_read
 * Ptimestamp_SE_read            Pa_timestamp_SE_read            Pe_timestamp_SE_read
 * Ptimestamp_CSE_read           Pa_timestamp_CSE_read           Pe_timestamp_CSE_read
 *
 * Pdate_FW_read                 Pa_date_FW_read                 Pe_date_FW_read
 * Pdate_read                    Pa_date_read                    Pe_date_read
 * Pdate_ME_read                 Pa_date_ME_read                 Pe_date_ME_read
 * Pdate_CME_read                Pa_date_CME_read                Pe_date_CME_read
 * Pdate_SE_read                 Pa_date_SE_read                 Pe_date_SE_read
 * Pdate_CSE_read                Pa_date_CSE_read                Pe_date_CSE_read
 *
 * Ptime_FW_read                 Pa_time_FW_read                 Pe_time_FW_read
 * Ptime_read                    Pa_time_read                    Pe_time_read
 * Ptime_ME_read                 Pa_time_ME_read                 Pe_time_ME_read
 * Ptime_CME_read                Pa_time_CME_read                Pe_time_CME_read
 * Ptime_SE_read                 Pa_time_SE_read                 Pe_time_SE_read
 * Ptime_CSE_read                Pa_time_CSE_read                Pe_time_CSE_read
 */
```

```
/*
 * Ptimestamp_explicit variants:
 *
 *    Converts ASCII/EBCDIC char date/time description into seconds
 *    since Midnight Jan 1, 1970.  Format-based parsing is based on
 *    libast's tmdate function.  Input format and input time zone are
 *    specified explicitly.  The format used controls whether input is
 *    date and time, just date, or just time.  Default output format
 *    is set via disc->out_formats.timestamp_explicit
 *
 * Pdate_explicit variants:
 *
 *    Like Ptimestamp_explicit, with explicit format and time zone
 *    argument.  INTENDED to be used for just date, but format
 *    determines which strings are accepted. If treated as a full timestamp
 *    (time in seconds since Midnight Jan 1, 1970), the result has a
 *    'time of day' component of 0 hours, 0 minutes, 0 seconds.
 *    Default output format is set via disc->out_formats.date_explicit
 *
 * Ptime_explicit variants:
 *
 *    Like Ptimestamp_explicit, with explicit format and time zone
 *    argument.  INTENDED to be used for just time, but format
 *    determines which strings are accepted.  The resulting
 *    time in seconds is just the contribution of the specified time of day,
 *    thus if treated as a full timestamp, the time would fall on Jan 1, 1970.
 *    Default output format is set via disc->out_formats.time_explicit
 */
```

```
/*
 * Ptimestamp variants:
 *
 *    Like Ptimestamp_explicit, but input format and input time zone
 *    are taken from disc->in_formats.timestamp and
 *    disc->in_time_zone.  INTENDED to be used for both a date and
 *    time, but format determines actual use.  Default output format
 *    is set via disc->out_formats.timestamp
 *
 * Pdate variants:
 *
 *    Like Pdate_explicit, but input format and input time zone are
 *    taken from disc->in_formats.date and disc->in_time_zone.
 *    INTENDED to be used for just date, but format determines which
 *    strings are accepted. If treated as a full timestamp (time in seconds
 *    since Midnight Jan 1, 1970), the result has a 'time of day' component
 *    of 0 hours, 0 minutes, 0 seconds.  Default output format is set
 *    via disc->out_formats.date
 *
 * Ptime variants:
 *
 *    Like Ptime_explicit, but input format and input time zone are
 *    taken from disc->in_formats.time and disc->in_time_zone.
 *    INTENDED to be used for just time, but format determines which
 *    strings are accepted.  The resulting time in seconds is just the
 *    contribution of the specified time of day, thus if treated as a
 *    full timestamp, the time would fall on Jan 1, 1970.
 *    Default output format is set via disc->out_formats.time
 *
 * Each of the types above corresponds to one of the Pstring variants.
 * In each case one specifies the extent of a 'string' in the input
 * that is to be converted to a Puint32 representing the date in
 * seconds since the epoch.  For the different date formats that are
 * supported, see the discussion of disc->in_formats in pads.h.
 *
 * If the current IO cursor position points to a valid date string:
 *   + Sets (*res_out) to the resulting date in seconds since the epoch
 *   + advances the IO cursor position to just after the last
 *     legal character in the date string
 *   + returns P_OK
 * Otherwise:
 *   + does not advance the IO cursor pos
 *   + returns P_ERR
 */
```

```
Perror_t Pa_timestamp_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, size_t width, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_timestamp_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pchar stopChar, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_timestamp_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_timestamp_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_timestamp_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_timestamp_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );
```

```
Perror_t Pa_date_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, size_t width, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_date_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pchar stopChar, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_date_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_date_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_date_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pa_date_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );
```

```
Perror_t Pa_time_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pa_time_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pa_time_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pa_time_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pa_time_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pa_time_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
Perror_t Pa_timestamp_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width
                  );

Perror_t Pa_timestamp_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar
                  );

Perror_t Pa_timestamp_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp
                  );

Perror_t Pa_timestamp_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp
                  );

Perror_t Pa_timestamp_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp
                  );

Perror_t Pa_timestamp_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp
                  );
```

```
Perror_t Pa_date_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  size_t width
                  );

Perror_t Pa_date_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pchar stopChar
                  );

Perror_t Pa_date_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *matchRegexp
                  );

Perror_t Pa_date_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *matchRegexp
                  );

Perror_t Pa_date_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *stopRegexp
                  );

Perror_t Pa_date_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *stopRegexp
                  );
```

```
Perror_t Pa_time_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  size_t width
                  );

Perror_t Pa_time_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pchar stopChar
                  );

Perror_t Pa_time_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *matchRegexp
                  );

Perror_t Pa_time_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *matchRegexp
                  );

Perror_t Pa_time_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *stopRegexp
                  );

Perror_t Pa_time_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *stopRegexp
                  );
```

```
Perror_t Pe_timestamp_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, size_t width, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_timestamp_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pchar stopChar, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_timestamp_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_timestamp_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_timestamp_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_timestamp_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );
```

```
Perror_t Pe_date_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, size_t width, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_date_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pchar stopChar, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_date_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_date_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_date_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, const char *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );

Perror_t Pe_date_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                   Tm_zone_t *tzone
                   );
```

```
Perror_t Pe_time_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pe_time_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pe_time_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pe_time_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pe_time_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pe_time_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
Perror_t Pe_timestamp_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width
                  );

Perror_t Pe_timestamp_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar
                  );

Perror_t Pe_timestamp_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp
                  );

Perror_t Pe_timestamp_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp
                  );

Perror_t Pe_timestamp_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp
                  );

Perror_t Pe_timestamp_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp
                  );
```

```
Perror_t Pe_date_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  size_t width
                  );

Perror_t Pe_date_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pchar stopChar
                  );

Perror_t Pe_date_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *matchRegexp
                  );

Perror_t Pe_date_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *matchRegexp
                  );

Perror_t Pe_date_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *stopRegexp
                  );

Perror_t Pe_date_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *stopRegexp
                  );
```

```
Perror_t Pe_time_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  size_t width
                  );

Perror_t Pe_time_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pchar stopChar
                  );

Perror_t Pe_time_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *matchRegexp
                  );

Perror_t Pe_time_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *matchRegexp
                  );

Perror_t Pe_time_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  const char *stopRegexp
                  );

Perror_t Pe_time_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pregexp_t *stopRegexp
                  );
```

```
Perror_t Ptimestamp_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptimestamp_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptimestamp_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptimestamp_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptimestamp_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptimestamp_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
Perror_t Pdate_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pdate_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pdate_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pdate_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pdate_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Pdate_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
Perror_t Ptime_explicit_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptime_explicit_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptime_explicit_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptime_explicit_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptime_explicit_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

Perror_t Ptime_explicit_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
Perror_t Ptimestamp_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width
                  );

Perror_t Ptimestamp_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                  Pchar stopChar
                  );

Perror_t Ptimestamp_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *matchRegexp
                  );

Perror_t Ptimestamp_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *matchRegexp
                  );

Perror_t Ptimestamp_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, const char *stopRegexp
                  );

Perror_t Ptimestamp_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, Pregexp_t *stopRegexp
                  );
```

```
Perror_t Pdate_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   size_t width
                   );

Perror_t Pdate_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   Pchar stopChar);

Perror_t Pdate_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   const char *matchRegexp
                   );

Perror_t Pdate_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   Pregexp_t *matchRegexp
                   );

Perror_t Pdate_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   const char *stopRegexp
                   );

Perror_t Pdate_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   Pregexp_t *stopRegexp
                   );
```

```
Perror_t Ptime_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   size_t width
                   );

Perror_t Ptime_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   Pchar stopChar);

Perror_t Ptime_ME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   const char *matchRegexp
                   );

Perror_t Ptime_CME_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   Pregexp_t *matchRegexp
                   );

Perror_t Ptime_SE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   const char *stopRegexp
                   );

Perror_t Ptime_CSE_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                   Pregexp_t *stopRegexp
                   );
```

```
/* ========================================================================
 * WRITE FUNCTIONS
 * DEFAULT                      ASCII                        EBCDIC
 * --------------------------  ---------------------------  ----------------------
 * Pdate_FW_write2io           Pa_date_FW_write2io          Pe_date_FW_write2io
 * Pdate_write2io              Pa_date_write2io             Pe_date_write2io
 * Pdate_ME_write2io           Pa_date_ME_write2io          Pe_date_ME_write2io
 * Pdate_CME_write2io          Pa_date_CME_write2io         Pe_date_CME_write2io
 * Pdate_SE_write2io           Pa_date_SE_write2io          Pe_date_SE_write2io
 * Pdate_CSE_write2io          Pa_date_CSE_write2io         Pe_date_CSE_write2io
 *
 * Pdate_FW_write2buf          Pa_date_FW_write2buf         Pe_date_FW_write2buf
 * Pdate_write2buf             Pa_date_write2buf            Pe_date_write2buf
 * Pdate_ME_write2buf          Pa_date_ME_write2buf         Pe_date_ME_write2buf
 * Pdate_CME_write2buf         Pa_date_CME_write2buf        Pe_date_CME_write2buf
 * Pdate_SE_write2buf          Pa_date_SE_write2buf         Pe_date_SE_write2buf
 * Pdate_CSE_write2buf         Pa_date_CSE_write2buf        Pe_date_CSE_write2buf
 */
```

```
/* Ptimestamp_explicit */

ssize_t Pa_timestamp_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_timestamp_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );




ssize_t Pa_timestamp_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
ssize_t Pa_timestamp_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf,
                    size_t buf_len, int *buf_full, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf,
                    size_t buf_len, int *buf_full, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_timestamp_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_timestamp_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_timestamp_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_timestamp_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_timestamp_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_timestamp_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_timestamp_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );




ssize_t Pe_timestamp_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, size_t width,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pchar stopChar,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_timestamp_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf,
                    size_t buf_len, int *buf_full, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf,
                    size_t buf_len, int *buf_full, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_timestamp_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_timestamp_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_timestamp_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_timestamp_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_timestamp_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Ptimestamp_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Ptimestamp_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, Pregexp_t *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, Pregexp_t *stopRegexp, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );




ssize_t Ptimestamp_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptimestamp_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
ssize_t Ptimestamp_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Ptimestamp_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Ptimestamp_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Ptimestamp_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *stopRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptimestamp_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                    );




/* Pdate_explicit */

ssize_t Pa_date_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_date_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );




ssize_t Pa_date_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
ssize_t Pa_date_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_date_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_date_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_date_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_date_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_date_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_date_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_date_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );




ssize_t Pe_date_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_date_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_date_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_date_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_date_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_date_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_date_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pdate_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pdate_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pdate_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pdate_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Pdate_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pdate_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pdate_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                   const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *stopRegexp, const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                   const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                   const char *format, Tm_zone_t *tzone
                   );
```

```
ssize_t Pdate_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Puint32 *d, const char *tag, int indent, size_t width,
                   const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, size_t width, const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pchar stopChar, const char *format,
                   Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                   const char *format, Tm_zone_t *tzone
                   );

ssize_t Pdate_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, const char *matchRegexp, const char *format,
                   Tm_zone_t *tzone
                   );
```

```
ssize_t Pdate_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Pdate_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pdate_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pdate_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *stopRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pdate_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                    );




/* Ptime_explicit */

ssize_t Pa_time_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_time_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );




ssize_t Pa_time_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
ssize_t Pa_time_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_time_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_time_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pa_time_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pa_time_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pa_time_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_time_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_time_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                  const char *format, Tm_zone_t *tzone
                  );




ssize_t Pe_time_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pchar stopChar,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp, const char *format,
                  Tm_zone_t *tzone
                  );
```

```
ssize_t Pe_time_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_time_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_time_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Pe_time_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Pe_time_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Pe_time_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Ptime_explicit_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptime_explicit_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width,
                  const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptime_explicit_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptime_explicit_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, Pchar stopChar, const char *format,
                  Tm_zone_t *tzone
                  );

ssize_t Ptime_explicit_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp, const char *format, Tm_zone_t *tzone
                  );

ssize_t Ptime_explicit_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp,
                  const char *format, Tm_zone_t *tzone
                  );
```

```
ssize_t Ptime_explicit_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );




ssize_t Ptime_explicit_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, size_t width,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Ptime_explicit_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp, const char *format,
                    Tm_zone_t *tzone
                    );
```

```
ssize_t Ptime_explicit_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pchar stopChar, const char *format,
                    Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Ptime_explicit_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp,
                    const char *format, Tm_zone_t *tzone
                    );
```

```
ssize_t Ptime_explicit_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pchar stopChar, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *stopRegexp, const char *format, Tm_zone_t *tzone
                    );

ssize_t Ptime_explicit_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp, const char *format, Tm_zone_t *tzone
                    );
```

```
/* Ptimestamp */

ssize_t Pa_timestamp_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    size_t width
                    );

ssize_t Pa_timestamp_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width
                    );

ssize_t Pa_timestamp_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pchar stopChar
                    );

ssize_t Pa_timestamp_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pchar stopChar
                    );

ssize_t Pa_timestamp_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp
                    );

ssize_t Pa_timestamp_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                    );
```

```
ssize_t Pa_timestamp_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *stopRegexp
                  );

ssize_t Pa_timestamp_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_timestamp_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, size_t width
                  );

ssize_t Pa_timestamp_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pa_timestamp_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, Pchar stopChar
                  );

ssize_t Pa_timestamp_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pchar stopChar
                  );

ssize_t Pa_timestamp_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, const char *matchRegexp
                  );

ssize_t Pa_timestamp_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *matchRegexp
                  );
```

```
ssize_t Pa_timestamp_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, const char *stopRegexp
                  );

ssize_t Pa_timestamp_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                  Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_timestamp_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pa_timestamp_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, size_t width
                  );

ssize_t Pa_timestamp_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_timestamp_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_timestamp_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );

ssize_t Pa_timestamp_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *matchRegexp
                  );
```

```
ssize_t Pa_timestamp_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pa_timestamp_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_timestamp_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width
                  );

ssize_t Pa_timestamp_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar
                  );

ssize_t Pa_timestamp_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp
                  );

ssize_t Pa_timestamp_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pa_timestamp_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp
                  );

ssize_t Pa_timestamp_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_timestamp_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width
                  );

ssize_t Pe_timestamp_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, size_t width
                  );

ssize_t Pe_timestamp_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar
                  );

ssize_t Pe_timestamp_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, Pchar stopChar
                  );

ssize_t Pe_timestamp_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp
                  );

ssize_t Pe_timestamp_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                  );
```

```
ssize_t Pe_timestamp_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pe_timestamp_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                  );

ssize_t Pe_timestamp_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *stopRegexp
                  );

ssize_t Pe_timestamp_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                  );

ssize_t Pe_timestamp_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pregexp_t *stopRegexp
                  );

ssize_t Pe_timestamp_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_timestamp_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_timestamp_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pe_timestamp_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pchar stopChar
                    );

ssize_t Pe_timestamp_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pchar stopChar
                    );

ssize_t Pe_timestamp_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *matchRegexp
                    );

ssize_t Pe_timestamp_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp
                    );
```

```
ssize_t Pe_timestamp_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pe_timestamp_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pe_timestamp_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *stopRegexp
                    );

ssize_t Pe_timestamp_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp
                    );

ssize_t Pe_timestamp_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Puint32 *d, const char *tag, int indent, Pregexp_t *stopRegexp
                    );

ssize_t Pe_timestamp_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_timestamp_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pe_timestamp_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, size_t width
                  );

ssize_t Pe_timestamp_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pe_timestamp_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                  );

ssize_t Pe_timestamp_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );

ssize_t Pe_timestamp_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *matchRegexp
                  );
```

```
ssize_t Pe_timestamp_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pe_timestamp_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pe_timestamp_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pe_timestamp_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pe_timestamp_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pe_timestamp_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_timestamp_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width
                  );

ssize_t Pe_timestamp_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar
                  );

ssize_t Pe_timestamp_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp
                  );

ssize_t Pe_timestamp_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pe_timestamp_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp
                  );

ssize_t Pe_timestamp_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp
                  );
```

```
ssize_t Ptimestamp_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width
                  );

ssize_t Ptimestamp_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, size_t width
                  );

ssize_t Ptimestamp_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar
                  );

ssize_t Ptimestamp_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, Pchar stopChar
                  );

ssize_t Ptimestamp_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp
                  );

ssize_t Ptimestamp_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                  );
```

```
ssize_t Ptimestamp_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp
                    );

ssize_t Ptimestamp_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                    );

ssize_t Ptimestamp_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp
                    );

ssize_t Ptimestamp_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                    );

ssize_t Ptimestamp_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp
                    );

ssize_t Ptimestamp_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Ptimestamp_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, size_t width
                    );

ssize_t Ptimestamp_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width
                    );

ssize_t Ptimestamp_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pchar stopChar
                    );

ssize_t Ptimestamp_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pchar stopChar
                    );

ssize_t Ptimestamp_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *matchRegexp
                    );

ssize_t Ptimestamp_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp
                    );
```

```
ssize_t Ptimestamp_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *matchRegexp
                    );

ssize_t Ptimestamp_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp
                    );

ssize_t Ptimestamp_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *stopRegexp
                    );

ssize_t Ptimestamp_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp
                    );

ssize_t Ptimestamp_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *stopRegexp
                    );

ssize_t Ptimestamp_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Ptimestamp_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Ptimestamp_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width
                    );

ssize_t Ptimestamp_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar
                    );

ssize_t Ptimestamp_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar
                    );

ssize_t Ptimestamp_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *matchRegexp
                    );

ssize_t Ptimestamp_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, const char *matchRegexp
                    );
```

```
ssize_t Ptimestamp_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Ptimestamp_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Ptimestamp_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Ptimestamp_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, const char *stopRegexp
                  );

ssize_t Ptimestamp_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );

ssize_t Ptimestamp_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Ptimestamp_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width
                  );

ssize_t Ptimestamp_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pchar stopChar
                  );

ssize_t Ptimestamp_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp
                  );

ssize_t Ptimestamp_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp
                  );

ssize_t Ptimestamp_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp
                  );

ssize_t Ptimestamp_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp
                  );
```

```
/* Pdate */

ssize_t Pa_date_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    size_t width
                    );

ssize_t Pa_date_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, size_t width
                    );

ssize_t Pa_date_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pchar stopChar
                    );

ssize_t Pa_date_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pchar stopChar
                    );

ssize_t Pa_date_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp
                    );

ssize_t Pa_date_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                    );









ssize_t Pa_date_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pa_date_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                    );

ssize_t Pa_date_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp
                    );

ssize_t Pa_date_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                    );

ssize_t Pa_date_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp
                    );

ssize_t Pa_date_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pa_date_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, size_t width
                   );

ssize_t Pa_date_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, size_t width
                   );

ssize_t Pa_date_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pchar stopChar
                   );

ssize_t Pa_date_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   Pchar stopChar
                   );

ssize_t Pa_date_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, const char *matchRegexp
                   );

ssize_t Pa_date_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, const char *matchRegexp
                   );
```

```
ssize_t Pa_date_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pregexp_t *matchRegexp
                   );

ssize_t Pa_date_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, Pregexp_t *matchRegexp
                   );

ssize_t Pa_date_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, const char *stopRegexp
                   );

ssize_t Pa_date_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, const char *stopRegexp
                   );

ssize_t Pa_date_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pregexp_t *stopRegexp
                   );

ssize_t Pa_date_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                   int indent, Pregexp_t *stopRegexp
                   );
```

```
ssize_t Pa_date_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pa_date_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pa_date_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_date_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_date_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );

ssize_t Pa_date_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );




ssize_t Pa_date_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pa_date_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pa_date_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pa_date_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pa_date_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pa_date_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_date_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width
                  );

ssize_t Pa_date_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                  Pbase_m *m, Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_date_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp
                  );

ssize_t Pa_date_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pa_date_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp
                  );

ssize_t Pa_date_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_date_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  size_t width
                  );

ssize_t Pe_date_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, size_t width
                  );

ssize_t Pe_date_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  Pchar stopChar
                  );

ssize_t Pe_date_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, Pchar stopChar
                  );

ssize_t Pe_date_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *matchRegexp
                  );

ssize_t Pe_date_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                  );
```

```
ssize_t Pe_date_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pe_date_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                    );

ssize_t Pe_date_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp
                    );

ssize_t Pe_date_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                    );

ssize_t Pe_date_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp
                    );

ssize_t Pe_date_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_date_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_date_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pe_date_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pchar stopChar
                    );

ssize_t Pe_date_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                    Pchar stopChar
                    );

ssize_t Pe_date_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *matchRegexp
                    );

ssize_t Pe_date_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp
                    );
```

```
ssize_t Pe_date_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pe_date_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pe_date_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, const char *stopRegexp
                  );

ssize_t Pe_date_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *stopRegexp
                  );

ssize_t Pe_date_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, Pregexp_t *stopRegexp
                  );

ssize_t Pe_date_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_date_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pe_date_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pe_date_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pe_date_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pe_date_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );

ssize_t Pe_date_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );
```

```
ssize_t Pe_date_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pe_date_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pe_date_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pe_date_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pe_date_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pe_date_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pe_date_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  size_t width
                  );

ssize_t Pe_date_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                  Pbase_m *m, Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                  );

ssize_t Pe_date_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *matchRegexp
                  );

ssize_t Pe_date_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *matchRegexp
                  );

ssize_t Pe_date_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  const char *stopRegexp
                  );

ssize_t Pe_date_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                  Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pdate_FW_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d,size_t width);

ssize_t Pdate_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, size_t width
                    );

ssize_t Pdate_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d,Pchar stopChar);

ssize_t Pdate_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pchar stopChar
                    );

ssize_t Pdate_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp
                    );

ssize_t Pdate_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                    );
```

```
ssize_t Pdate_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pdate_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                    );

ssize_t Pdate_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp
                    );

ssize_t Pdate_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                    );

ssize_t Pdate_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp
                    );

ssize_t Pdate_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pdate_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, size_t width
                   );

ssize_t Pdate_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   size_t width
                   );

ssize_t Pdate_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pchar stopChar
                   );

ssize_t Pdate_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   Pchar stopChar
                   );

ssize_t Pdate_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, const char *matchRegexp
                   );

ssize_t Pdate_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   const char *matchRegexp
                   );
```

```
ssize_t Pdate_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pregexp_t *matchRegexp
                   );

ssize_t Pdate_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   Pregexp_t *matchRegexp
                   );

ssize_t Pdate_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, const char *stopRegexp
                   );

ssize_t Pdate_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   const char *stopRegexp
                   );

ssize_t Pdate_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pregexp_t *stopRegexp
                   );

ssize_t Pdate_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   Pregexp_t *stopRegexp
                   );
```

```
ssize_t Pdate_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pdate_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pdate_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pdate_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pdate_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );

ssize_t Pdate_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );




ssize_t Pdate_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pdate_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *matchRegexp
                  );

ssize_t Pdate_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pdate_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *stopRegexp
                  );

ssize_t Pdate_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );

ssize_t Pdate_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pdate_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, size_t width
                    );

ssize_t Pdate_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                    );

ssize_t Pdate_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, const char *matchRegexp
                    );

ssize_t Pdate_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pdate_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, const char *stopRegexp
                    );

ssize_t Pdate_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp
                    );
```

```
/* Ptime */

ssize_t Pa_time_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    size_t width
                    );

ssize_t Pa_time_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, size_t width
                    );

ssize_t Pa_time_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pchar stopChar
                    );

ssize_t Pa_time_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pchar stopChar
                    );

ssize_t Pa_time_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp
                    );

ssize_t Pa_time_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                    );
```

```
ssize_t Pa_time_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pa_time_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                    );

ssize_t Pa_time_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp
                    );

ssize_t Pa_time_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                    );

ssize_t Pa_time_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp
                    );

ssize_t Pa_time_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pa_time_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_time_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_time_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pchar stopChar
                    );

ssize_t Pa_time_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                    Pchar stopChar
                    );

ssize_t Pa_time_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *matchRegexp
                    );

ssize_t Pa_time_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp
                    );
```

```
ssize_t Pa_time_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pa_time_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pregexp_t *matchRegexp
                  );

ssize_t Pa_time_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, const char *stopRegexp
                  );

ssize_t Pa_time_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, const char *stopRegexp
                  );

ssize_t Pa_time_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                  const char *tag, int indent, Pregexp_t *stopRegexp
                  );

ssize_t Pa_time_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                  int indent, Pregexp_t *stopRegexp
                  );
```

```
ssize_t Pa_time_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pa_time_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, size_t width
                  );

ssize_t Pa_time_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_time_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Pchar stopChar
                  );

ssize_t Pa_time_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );

ssize_t Pa_time_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, const char *matchRegexp
                  );
```

```
ssize_t Pa_time_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *matchRegexp
                    );

ssize_t Pa_time_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                    );

ssize_t Pa_time_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *stopRegexp
                    );

ssize_t Pa_time_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *stopRegexp
                    );

ssize_t Pa_time_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *stopRegexp
                    );

ssize_t Pa_time_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pa_time_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width
                    );

ssize_t Pa_time_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                    );

ssize_t Pa_time_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *matchRegexp
                    );

ssize_t Pa_time_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pa_time_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *stopRegexp
                    );

ssize_t Pa_time_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_time_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    size_t width
                    );

ssize_t Pe_time_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, size_t width
                    );

ssize_t Pe_time_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pchar stopChar
                    );

ssize_t Pe_time_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pchar stopChar
                    );

ssize_t Pe_time_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp
                    );

ssize_t Pe_time_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                    );
```

```
ssize_t Pe_time_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *stopRegexp
                    );

ssize_t Pe_time_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                    );

ssize_t Pe_time_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    Pregexp_t *stopRegexp
                    );

ssize_t Pe_time_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_time_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_time_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pe_time_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pchar stopChar
                    );

ssize_t Pe_time_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                    Pchar stopChar
                    );

ssize_t Pe_time_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *matchRegexp
                    );

ssize_t Pe_time_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *matchRegexp
                    );
```

```
ssize_t Pe_time_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *stopRegexp
                    );

ssize_t Pe_time_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, const char *stopRegexp
                    );

ssize_t Pe_time_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *stopRegexp
                    );

ssize_t Pe_time_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *d, const char *tag,
                    int indent, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_time_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Pe_time_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Pe_time_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar
                    );

ssize_t Pe_time_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar
                    );

ssize_t Pe_time_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *matchRegexp
                    );

ssize_t Pe_time_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *matchRegexp
                    );




ssize_t Pe_time_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *stopRegexp
                    );

ssize_t Pe_time_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *stopRegexp
                    );

ssize_t Pe_time_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *stopRegexp
                    );

ssize_t Pe_time_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Pe_time_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width
                    );

ssize_t Pe_time_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                    );

ssize_t Pe_time_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *matchRegexp
                    );

ssize_t Pe_time_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp
                    );

ssize_t Pe_time_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    const char *stopRegexp
                    );

ssize_t Pe_time_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp
                    );
```

```
ssize_t Ptime_FW_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d,size_t width);

ssize_t Ptime_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, size_t width
                    );

ssize_t Ptime_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *d,Pchar stopChar);

ssize_t Ptime_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, Pchar stopChar
                    );

ssize_t Ptime_ME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *matchRegexp
                    );

ssize_t Ptime_ME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *matchRegexp
                    );
```

```
ssize_t Ptime_CME_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   Pregexp_t *matchRegexp
                   );

ssize_t Ptime_CME_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, Pregexp_t *matchRegexp
                   );

ssize_t Ptime_SE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *stopRegexp
                   );

ssize_t Ptime_SE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *stopRegexp
                   );

ssize_t Ptime_CSE_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   Pregexp_t *stopRegexp
                   );

ssize_t Ptime_CSE_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, Pregexp_t *stopRegexp
                   );
```

```
ssize_t Ptime_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, size_t width
                   );

ssize_t Ptime_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   size_t width
                   );

ssize_t Ptime_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, Pchar stopChar
                   );

ssize_t Ptime_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   Pchar stopChar
                   );

ssize_t Ptime_ME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                   const char *tag, int indent, const char *matchRegexp
                   );

ssize_t Ptime_ME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                   const char *matchRegexp
                   );
```

```
ssize_t Ptime_CME_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *matchRegexp
                    );

ssize_t Ptime_CME_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                    Pregexp_t *matchRegexp
                    );

ssize_t Ptime_SE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, const char *stopRegexp
                    );

ssize_t Ptime_SE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                    const char *stopRegexp
                    );

ssize_t Ptime_CSE_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *d,
                    const char *tag, int indent, Pregexp_t *stopRegexp
                    );

ssize_t Ptime_CSE_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *d, const char *tag, int indent,
                    Pregexp_t *stopRegexp
                    );
```

```
ssize_t Ptime_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Ptime_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Ptime_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar
                    );

ssize_t Ptime_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pchar stopChar
                    );

ssize_t Ptime_ME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *matchRegexp
                    );

ssize_t Ptime_ME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *matchRegexp
                    );
```

```
ssize_t Ptime_CME_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *matchRegexp
                    );

ssize_t Ptime_CME_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *matchRegexp
                    );

ssize_t Ptime_SE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *stopRegexp
                    );

ssize_t Ptime_SE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, const char *stopRegexp
                    );

ssize_t Ptime_CSE_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *stopRegexp
                    );

ssize_t Ptime_CSE_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Pregexp_t *stopRegexp
                    );
```

```
ssize_t Ptime_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, size_t width
                    );

ssize_t Ptime_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, Pchar stopChar
                    );

ssize_t Ptime_ME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, const char *matchRegexp
                    );

ssize_t Ptime_CME_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *matchRegexp
                    );

ssize_t Ptime_SE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Puint32 *rep, const char *stopRegexp
                    );

ssize_t Ptime_CSE_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Pregexp_t *stopRegexp
                    );
```

B.6 Integers: Character encodings

```
/* ============================================================================
 * ASCII STRING TO INTEGER READ FUNCTIONS
 *
 * An ASCII representation of an integer value (a string of digits in
 * [0-9]) is assumed to be at the current cursor position, where if
 * the target type is a signed type a leading - or + is allowed and if
 * unsigned a leading + is allowed.  If (pads->disc->flags &
 * P_WSPACE_OK), leading white space is skipped, otherwise leading
 * white space causes an error.  Thus, the string to be converted
 * consists of: optional white space, optional +/-, and all
 * consecutive digits (first nondigit marks end).
 *
 * RETURN VALUE: Perror_t
 *
 * Upon success, P_OK returned:
 *   + the IO cursor is advanced to just beyond the last digit
 *   + if P_Test_NotIngore(*m), the out param is assigned a value
 *
 * P_ERR is returned on error.
 * Cursor advancement/err settings for different error cases:
 */
```

```
/* (1) If IO cursor is at EOF
 *     + pd->loc.b/e set to EOF 'location'
 *     + IO cursor remains at EOF
 *     + if P_Test_NotIgnore(*), pd->errCode set to P_AT_EOF,
 *         pd->nerr set to 1, and an error is reported
 * (2a) There is leading white space and
 *        (pads->disc->flags & P_WSPACE_OK) is not set
 * (2b) The target is unsigned and the first char is a -
 * (2c) The first character is not a +, -, or in [0-9]
 * (2d) First character is allowable + or -, following by a char that
 *       is not a digit
 * For the above 4 cases:
 *     + pd->loc.b/e set to the IO cursor position.
 *     + IO cursor is not advanced
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_INVALID_A_NUM,
 *         pd->nerr set to 1, and an error is reported
 * (3) A valid ASCII integer string is found, but it describes
 *     an integer that does not fit in the specified target type
 *     + pd->loc.b/e set to elt/char position of start and
 *       end of the ASCII integer
 *     + IO cursor is advanced just beyond the last digit
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 */
```

```
Perror_t Pa_int8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint8 *res_out);

Perror_t Pa_int16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint16 *res_out);

Perror_t Pa_int32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint32 *res_out);

Perror_t Pa_int64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint64 *res_out);

Perror_t Pa_uint8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint8 *res_out);

Perror_t Pa_uint16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint16 *res_out);

Perror_t Pa_uint32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);

Perror_t Pa_uint64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint64 *res_out);
```

```
/*
 * Fixed-width ASCII integer read functions:
 *    Like the above, only a fixed width in input characters is
 *    specified, and only those characters are examined.  E.g., input
 *    '11112222' could be used to read two fixed-width ASCII integers
 *    of width 4.
 *
 * N.B. The APIs require width > 0.  If width <= 0 is given, an
 * immediate error return occurs, without setting pd's location or
 * error code.
 */
```

```
/* Other differences from the variable-width read functions:
 *
 * 1. It is an error if the entire specified width is not an integer,
 *    e.g., for fixed width 4, input '111|' is an error
 *
 * 2. (pads->disc->flags & P_WSPACE_OK) indicates whether leading OR
 *    trailing spaces are OK, e.g., for fixed width 4, input ' 1 ' is not
 *    an error is wpace_ok is 1 (trailing white space is not an issue for
 *    variable-width routines)
 *
 * 3. If the specified width is available, it is always consumed, even
 *    if there is an error.  In this case
 *      + pd->loc.b/e is set to the first/last char of the fixed-width field
 *      + if P_Test_NotIgnore(*m), an error code is set,
 *         pd->nerr set to 1, and an error is reported
 *
 *    If the specified width is *not* available (EOR/EOF hit):
 *      + pd->loc.b/e set to elt/char position of start/end of
 *          the 'too small' field
 *      + IO cursor is not advanced
 *      + if P_Test_NotIgnore(*m), pd->errCode set to P_WIDTH_NOT_AVAILABLE,
 *         pd->nerr set to 1, and an error is reported
 */
```

```
Perror_t Pa_int8_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                    size_t width
                    );

Perror_t Pa_int16_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                    size_t width
                    );

Perror_t Pa_int32_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    size_t width
                    );

Perror_t Pa_int64_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                    size_t width
                    );
```

```
Perror_t Pa_uint8_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                    size_t width
                    );

Perror_t Pa_uint16_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Puint16 *res_out, size_t width
                    );

Perror_t Pa_uint32_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Puint32 *res_out, size_t width
                    );

Perror_t Pa_uint64_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Puint64 *res_out, size_t width
                    );
```

```
/* ================================================================================
 * EBCDIC STRING TO INTEGER READ FUNCTIONS
 *
 * These functions are just like their ASCII counterparts; the only
 * difference is the integers are encoding using EBCDIC string data.
 * The error codes used are also the same,
 * except that error code P_INVALID_E_NUM is used rather
 * than P_INVALID_A_NUM
 */
```

```
Perror_t Pe_int8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint8 *res_out);

Perror_t Pe_int16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint16 *res_out);

Perror_t Pe_int32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint32 *res_out);

Perror_t Pe_int64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint64 *res_out);

Perror_t Pe_uint8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint8 *res_out);

Perror_t Pe_uint16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint16 *res_out);

Perror_t Pe_uint32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);

Perror_t Pe_uint64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint64 *res_out);
```

```
Perror_t Pe_int8_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                  size_t width
                  );

Perror_t Pe_int16_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                  size_t width
                  );

Perror_t Pe_int32_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                  size_t width
                  );

Perror_t Pe_int64_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                  size_t width
                  );
```

```
Perror_t Pe_uint8_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                  size_t width
                  );

Perror_t Pe_uint16_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint16 *res_out, size_t width
                  );

Perror_t Pe_uint32_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint32 *res_out, size_t width
                  );

Perror_t Pe_uint64_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                  Puint64 *res_out, size_t width
                  );
```

```
/* ============================================================================
 * DEFAULT STRING TO INTEGER READ FUNCTIONS
 *
 * These functions select the appropriate ASCII or EBCDIC string to integer
 * function based on pads->disc->def_charset.
 *
 * Example: the call
 *
 *     Pint8_read(pads, &m, &ed, *res)
 *
 * is converted to one of these forms:
 *
 *     Pa_int8_read(pads, &m, &ed, *res)
 *     Pe_int8_read(pads, &m, &ed, *res)
 *     etc.
 */
```

```
Perror_t Pint8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint8 *res_out);

Perror_t Pint16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint16 *res_out);

Perror_t Pint32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint32 *res_out);

Perror_t Pint64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint64 *res_out);

Perror_t Puint8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint8 *res_out);

Perror_t Puint16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint16 *res_out);

Perror_t Puint32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);

Perror_t Puint64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint64 *res_out);
```

```
Perror_t Pint8_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                   size_t width
                   );

Perror_t Pint16_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                   size_t width
                   );

Perror_t Pint32_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                   size_t width
                   );

Perror_t Pint64_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                   size_t width
                   );
```

```
Perror_t Puint8_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                    size_t width
                    );

Perror_t Puint16_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint16 *res_out,
                    size_t width
                    );

Perror_t Puint32_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                    size_t width
                    );

Perror_t Puint64_FW_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint64 *res_out,
                    size_t width
                    );
```

```
/* ================================================================================
 * WRITE
 */

/* ================================================================================
 * INTEGER/FPOINT WRITE FUNCTIONS
 *
 * For each integer or fpoint read function there is a corresponding write2io
 * function and a corresponding write2buf function which output the specified
 * value in a format that will allow the corresponding read function to
 * successfully read the value.
 *
 * For example, if a Pint8 is written using Pe_int8_write2io, the bytes
 * that were output can be read back into a Pint8 using Pe_int8_read.
 *
 * All write functions take an Sfio_t* stream pointer (the stream to write to),
 * a parse descriptor pd, and a pointer to the value to be written.  Some also take
 * additional arguments, such as num_digits.  All return an integer.
 *
 * If pd->errCode is either P_NO_ERR or P_USER_CONSTRAINT_VIOLATIONS then
 * the value is assumed to have been filled in, and it is the value written.
 * For other error codes, the value is assumed to *not* have been filled in,
 * and an error value is written.  See the Default Error Value discussion above
 * for the set of default error values and details on how to override them.
 *
 * If the write succeeds, the return value is the number of bytes written.
 * If it fails, -1 is returned, and no bytes are written to the stream.
 */
```

```
ssize_t Pa_int8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint8 *val);

ssize_t Pa_int16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint16 *val);

ssize_t Pa_int32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint32 *val);

ssize_t Pa_int64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint64 *val);
```

```
ssize_t Pa_uint8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint8 *val);

ssize_t Pa_uint16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint16 *val);

ssize_t Pa_uint32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *val);

ssize_t Pa_uint64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint64 *val);
```

```
ssize_t Pa_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pa_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pa_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pa_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Pa_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pa_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pa_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pa_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Pe_int8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint8 *val);

ssize_t Pe_int16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint16 *val);

ssize_t Pe_int32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint32 *val);

ssize_t Pe_int64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint64 *val);
```

```
ssize_t Pe_uint8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint8 *val);

ssize_t Pe_uint16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint16 *val);

ssize_t Pe_uint32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *val);

ssize_t Pe_uint64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint64 *val);
```

```
ssize_t Pe_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pe_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pe_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pe_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Pe_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pe_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pe_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pe_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Pa_int8_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    size_t width
                    );

ssize_t Pa_int16_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    size_t width
                    );

ssize_t Pa_int32_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    size_t width
                    );

ssize_t Pa_int64_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    size_t width
                    );
```

```
ssize_t Pa_uint8_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    size_t width
                    );

ssize_t Pa_uint16_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    size_t width
                    );

ssize_t Pa_uint32_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    size_t width
                    );

ssize_t Pa_uint64_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    size_t width
                    );
```

```
ssize_t Pa_int8_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_int16_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_int32_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_int64_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, size_t width
                    );
```

```
ssize_t Pa_uint8_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_uint16_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_uint32_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pa_uint64_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, size_t width
                    );
```

```
ssize_t Pe_int8_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    size_t width
                    );

ssize_t Pe_int16_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    size_t width
                    );

ssize_t Pe_int32_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    size_t width
                    );

ssize_t Pe_int64_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    size_t width
                    );
```

```
ssize_t Pe_uint8_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    size_t width
                    );

ssize_t Pe_uint16_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    size_t width
                    );

ssize_t Pe_uint32_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    size_t width
                    );

ssize_t Pe_uint64_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    size_t width
                    );
```

```
ssize_t Pe_int8_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_int16_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_int32_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_int64_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, size_t width
                    );
```

```
ssize_t Pe_uint8_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_uint16_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_uint32_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pe_uint64_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, size_t width
                    );
```

```
ssize_t Pa_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val
                    );

ssize_t Pa_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val
                    );

ssize_t Pa_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val
                    );

ssize_t Pa_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val
                    );
```

```
ssize_t Pa_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val
                    );

ssize_t Pa_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val
                    );

ssize_t Pa_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val
                    );

ssize_t Pa_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val
                    );
```

```
ssize_t Pa_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent
                    );

ssize_t Pa_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, const char *tag, int indent
                    );

ssize_t Pa_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, const char *tag, int indent
                    );

ssize_t Pa_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, const char *tag, int indent
                    );
```

```
ssize_t Pa_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, const char *tag, int indent
                    );

ssize_t Pa_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, const char *tag, int indent
                    );

ssize_t Pa_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, const char *tag, int indent
                    );

ssize_t Pa_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, const char *tag, int indent
                    );
```

```
ssize_t Pe_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val
                    );

ssize_t Pe_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val
                    );

ssize_t Pe_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val
                    );

ssize_t Pe_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val
                    );
```

```
ssize_t Pe_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val
                    );

ssize_t Pe_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val
                    );

ssize_t Pe_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val
                    );

ssize_t Pe_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val
                    );
```

```
ssize_t Pe_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent
                    );

ssize_t Pe_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, const char *tag, int indent
                    );

ssize_t Pe_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, const char *tag, int indent
                    );

ssize_t Pe_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, const char *tag, int indent
                    );
```

```
ssize_t Pe_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, const char *tag, int indent
                    );

ssize_t Pe_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, const char *tag, int indent
                    );

ssize_t Pe_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, const char *tag, int indent
                    );

ssize_t Pe_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, const char *tag, int indent
                    );
```

```
ssize_t Pa_int8_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, size_t width
                    );

ssize_t Pa_int16_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, size_t width
                    );

ssize_t Pa_int32_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, size_t width
                    );

ssize_t Pa_int64_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, size_t width
                    );
```

```
ssize_t Pa_uint8_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, size_t width
                    );

ssize_t Pa_uint16_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, size_t width
                    );

ssize_t Pa_uint32_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, size_t width
                    );

ssize_t Pa_uint64_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, size_t width
                    );
```

```
ssize_t Pa_int8_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint8 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_int16_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint16 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_int32_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_int64_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint64 *val, const char *tag,
                    int indent, size_t width
                    );




ssize_t Pa_uint8_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint8 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_uint16_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_uint32_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Pa_uint64_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                    int indent, size_t width
                    );




ssize_t Pe_int8_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, size_t width
                    );

ssize_t Pe_int16_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, size_t width
                    );

ssize_t Pe_int32_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, size_t width
                    );

ssize_t Pe_int64_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, size_t width
                    );
```

```
ssize_t Pe_uint8_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint8 *val, size_t width
                  );

ssize_t Pe_uint16_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint16 *val, size_t width
                  );

ssize_t Pe_uint32_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *val, size_t width
                  );

ssize_t Pe_uint64_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint64 *val, size_t width
                  );
```

```
ssize_t Pe_int8_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pint8 *val, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pe_int16_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pint16 *val, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pe_int32_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pint32 *val, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pe_int64_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pint64 *val, const char *tag,
                  int indent, size_t width
                  );
```

```
ssize_t Pe_uint8_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint8 *val, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pe_uint16_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pe_uint32_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                  int indent, size_t width
                  );

ssize_t Pe_uint64_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                  int indent, size_t width
                  );
```

```
ssize_t Pa_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint8 *rep
                   );

ssize_t Pa_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint8 *rep
                   );

ssize_t Pa_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint16 *rep
                   );

ssize_t Pa_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint16 *rep
                   );

ssize_t Pa_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint32 *rep
                   );

ssize_t Pa_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint32 *rep
                   );

ssize_t Pa_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint64 *rep
                   );

ssize_t Pa_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint64 *rep
                   );
```

```
ssize_t Pa_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep
                  );

ssize_t Pa_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep
                  );

ssize_t Pa_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep
                  );

ssize_t Pa_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep
                  );

ssize_t Pa_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep
                  );

ssize_t Pa_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep
                  );

ssize_t Pa_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep
                  );

ssize_t Pa_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep
                  );
```

```
ssize_t Pa_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint8 *rep
                   );

ssize_t Pa_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint16 *rep
                   );

ssize_t Pa_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint32 *rep
                   );

ssize_t Pa_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint64 *rep
                   );

ssize_t Pa_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Puint8 *rep
                   );

ssize_t Pa_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep
                   );

ssize_t Pa_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep
                   );

ssize_t Pa_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep
                   );
```

```
ssize_t Pa_int8_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, size_t width
                    );

ssize_t Pa_int8_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, size_t width
                    );

ssize_t Pa_int16_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, size_t width
                    );

ssize_t Pa_int16_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Pint16 *rep, size_t width
                    );

ssize_t Pa_int32_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, size_t width
                    );

ssize_t Pa_int32_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Pint32 *rep, size_t width
                    );

ssize_t Pa_int64_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, size_t width
                    );

ssize_t Pa_int64_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Pint64 *rep, size_t width
                    );
```

```
ssize_t Pa_uint8_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, size_t width
                    );

ssize_t Pa_uint8_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint8 *rep, size_t width
                    );

ssize_t Pa_uint16_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint16 *rep, size_t width
                    );

ssize_t Pa_uint16_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint16 *rep, size_t width
                    );

ssize_t Pa_uint32_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Pa_uint32_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width
                    );

ssize_t Pa_uint64_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint64 *rep, size_t width
                    );

ssize_t Pa_uint64_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint64 *rep, size_t width
                    );
```

```
ssize_t Pa_int8_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint8 *rep,
                    size_t width
                    );

ssize_t Pa_int16_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                    size_t width
                    );

ssize_t Pa_int32_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                    size_t width
                    );

ssize_t Pa_int64_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                    size_t width
                    );

ssize_t Pa_uint8_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    size_t width
                    );

ssize_t Pa_uint16_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    size_t width
                    );

ssize_t Pa_uint32_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width
                    );

ssize_t Pa_uint64_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    size_t width
                    );
```

```
ssize_t Pe_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint8 *rep
                   );

ssize_t Pe_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint8 *rep
                   );

ssize_t Pe_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint16 *rep
                   );

ssize_t Pe_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint16 *rep
                   );

ssize_t Pe_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint32 *rep
                   );

ssize_t Pe_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint32 *rep
                   );

ssize_t Pe_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint64 *rep
                   );

ssize_t Pe_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint64 *rep
                   );
```

```
ssize_t Pe_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep
                  );

ssize_t Pe_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep
                  );

ssize_t Pe_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep
                  );

ssize_t Pe_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep
                  );

ssize_t Pe_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep
                  );

ssize_t Pe_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep
                  );

ssize_t Pe_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep
                  );

ssize_t Pe_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep
                  );
```

```
ssize_t Pe_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint8 *rep
                   );

ssize_t Pe_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint16 *rep
                   );

ssize_t Pe_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint32 *rep
                   );

ssize_t Pe_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint64 *rep
                   );

ssize_t Pe_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Puint8 *rep
                   );

ssize_t Pe_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep
                   );

ssize_t Pe_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep
                   );

ssize_t Pe_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep
                   );
```

```
ssize_t Pe_int8_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint8 *rep, size_t width
                   );

ssize_t Pe_int8_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint8 *rep, size_t width
                   );

ssize_t Pe_int16_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint16 *rep, size_t width
                   );

ssize_t Pe_int16_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                   Pbase_pd *pd, Pint16 *rep, size_t width
                   );

ssize_t Pe_int32_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint32 *rep, size_t width
                   );

ssize_t Pe_int32_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                   Pbase_pd *pd, Pint32 *rep, size_t width
                   );

ssize_t Pe_int64_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Pint64 *rep, size_t width
                   );

ssize_t Pe_int64_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                   Pbase_pd *pd, Pint64 *rep, size_t width
                   );
```

```
ssize_t Pe_uint8_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, size_t width
                    );

ssize_t Pe_uint8_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint8 *rep, size_t width
                    );

ssize_t Pe_uint16_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint16 *rep, size_t width
                    );

ssize_t Pe_uint16_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint16 *rep, size_t width
                    );

ssize_t Pe_uint32_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Pe_uint32_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, size_t width
                    );

ssize_t Pe_uint64_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint64 *rep, size_t width
                    );

ssize_t Pe_uint64_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint64 *rep, size_t width
                    );
```

```
ssize_t Pe_int8_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint8 *rep,
                    size_t width
                    );

ssize_t Pe_int16_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                    size_t width
                    );

ssize_t Pe_int32_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                    size_t width
                    );

ssize_t Pe_int64_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                    size_t width
                    );

ssize_t Pe_uint8_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    size_t width
                    );

ssize_t Pe_uint16_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    size_t width
                    );

ssize_t Pe_uint32_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width
                    );

ssize_t Pe_uint64_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    size_t width
                    );
```

```
ssize_t Pint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint8 *rep
                      );

ssize_t Pint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint8 *rep
                      );

ssize_t Pint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint16 *rep
                      );

ssize_t Pint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint16 *rep
                      );

ssize_t Pint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint32 *rep
                      );

ssize_t Pint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint32 *rep
                      );

ssize_t Pint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint64 *rep
                      );

ssize_t Pint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                      int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                      Pint64 *rep
                      );
```

```
ssize_t Puint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint8 *rep
                   );

ssize_t Puint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint8 *rep
                   );

ssize_t Puint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint16 *rep
                   );

ssize_t Puint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint16 *rep
                   );

ssize_t Puint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint32 *rep
                   );

ssize_t Puint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint32 *rep
                   );

ssize_t Puint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint64 *rep
                   );

ssize_t Puint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint64 *rep
                   );
```

```
ssize_t Pint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Pint8 *rep
                     );

ssize_t Pint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Pint16 *rep
                     );

ssize_t Pint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Pint32 *rep
                     );

ssize_t Pint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Pint64 *rep
                     );

ssize_t Puint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Puint8 *rep
                     );

ssize_t Puint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Puint16 *rep
                     );

ssize_t Puint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Puint32 *rep
                     );

ssize_t Puint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                     Pbase_m *m, Pbase_pd *pd, Puint64 *rep
                     );
```

```
ssize_t Pint8_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, size_t width
                    );

ssize_t Pint8_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, size_t width
                    );

ssize_t Pint16_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, size_t width
                    );

ssize_t Pint16_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, size_t width
                    );

ssize_t Pint32_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, size_t width
                    );

ssize_t Pint32_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, size_t width
                    );

ssize_t Pint64_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, size_t width
                    );

ssize_t Pint64_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, size_t width
                    );
```

```
ssize_t Puint8_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, size_t width
                    );

ssize_t Puint8_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, size_t width
                    );

ssize_t Puint16_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint16 *rep, size_t width
                    );

ssize_t Puint16_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint16 *rep, size_t width
                    );

ssize_t Puint32_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Puint32_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, size_t width
                    );

ssize_t Puint64_FW_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint64 *rep, size_t width
                    );

ssize_t Puint64_FW_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint64 *rep, size_t width
                    );
```

```
ssize_t Pint8_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                    Pbase_m *m, Pbase_pd *pd, Pint8 *rep, size_t width
                    );

ssize_t Pint16_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                    size_t width
                    );

ssize_t Pint32_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                    size_t width
                    );

ssize_t Pint64_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                    size_t width
                    );

ssize_t Puint8_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    size_t width
                    );

ssize_t Puint16_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    size_t width
                    );

ssize_t Puint32_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    size_t width
                    );

ssize_t Puint64_FW_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    size_t width
                    );
```

```
ssize_t Pint8_FW_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint8 *val,size_t width);

ssize_t Pint16_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    size_t width
                    );

ssize_t Pint32_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    size_t width
                    );

ssize_t Pint64_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    size_t width
                    );

ssize_t Puint8_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    size_t width
                    );

ssize_t Puint16_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    size_t width
                    );

ssize_t Puint32_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    size_t width
                    );

ssize_t Puint64_FW_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    size_t width
                    );
```

```
ssize_t Pint8_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, size_t width
                    );

ssize_t Pint16_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, size_t width
                    );

ssize_t Pint32_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, size_t width
                    );

ssize_t Pint64_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, size_t width
                    );
```

```
ssize_t Puint8_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, size_t width
                    );

ssize_t Puint16_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, size_t width
                    );

ssize_t Puint32_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, size_t width
                    );

ssize_t Puint64_FW_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, size_t width
                    );
```

```
ssize_t Pint8_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pint16_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pint32_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Pint64_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, size_t width
                    );
```

```
ssize_t Puint8_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Puint16_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Puint32_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, size_t width
                    );

ssize_t Puint64_FW_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, size_t width
                    );
```

```
ssize_t Pint8_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent,
                    size_t width
                    );

ssize_t Pint16_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, const char *tag, int indent,
                    size_t width
                    );

ssize_t Pint32_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, const char *tag, int indent,
                    size_t width
                    );

ssize_t Pint64_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, const char *tag, int indent,
                    size_t width
                    );
```

```
ssize_t Puint8_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, const char *tag, int indent,
                    size_t width
                    );

ssize_t Puint16_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Puint32_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                    int indent, size_t width
                    );

ssize_t Puint64_FW_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                    int indent, size_t width
                    );
```

```
ssize_t Pint8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint8 *val);

ssize_t Pint16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint16 *val);

ssize_t Pint32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint32 *val);

ssize_t Pint64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint64 *val);
```

```
ssize_t Puint8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint8 *val);

ssize_t Puint16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint16 *val);

ssize_t Puint32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *val);

ssize_t Puint64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint64 *val);
```

```
ssize_t Pint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val
                    );

ssize_t Pint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val
                    );

ssize_t Pint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val
                    );

ssize_t Pint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val
                    );
```

```
ssize_t Puint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val
                    );

ssize_t Puint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val
                    );

ssize_t Puint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val
                    );

ssize_t Puint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val
                    );
```

```
ssize_t Pint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Puint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent
                    );

ssize_t Puint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent
                    );

ssize_t Puint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent
                    );

ssize_t Puint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Pint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent
                    );

ssize_t Pint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, const char *tag, int indent
                    );

ssize_t Pint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, const char *tag, int indent
                    );

ssize_t Pint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, const char *tag, int indent
                    );
```

```
ssize_t Puint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, const char *tag, int indent
                    );

ssize_t Puint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, const char *tag, int indent
                    );

ssize_t Puint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, const char *tag, int indent
                    );

ssize_t Puint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, const char *tag, int indent
                    );
```

B.7 Integers: Raw binary encoding

```
/* ================================================================================
 * READ
 */

/* ================================================================================
 * COMMON WIDTH BINARY INTEGER READ FUNCTIONS
 *
 * These functions parse signed or unsigned binary integers of common
 * bit widths (8, 16, 32, and 64 bit widths).  Whether bytes are
 * reversed is controlled by the endian-ness of the machine
 * (determined automatically) and pads->disc->d_endian. If they
 * differ, byte order is reversed in the in-memory representation,
 * otherwise it is not.
 *
 * A good way to set the d_endian value in a machine-independent way
 * is to use the Pendian annotation with the first multi-byte binary
 * integer field that appears in the data.  For example, this header
 * definition:
 *
 * Pstruct header {
 *    Pendian Pb_uint16 version : version < 10;
 *    ..etc..
 * };
 *
 * indicates the first value is a 2-byte unsigned binary integer,
 * version, whose value should be less than 10.  The Pendian
 * annotation indicates that there should be two attempts at reading
 * the version field: once with the current pads->disc->d_endian
 * setting, and (if the read fails) once with the opposite
 * pads->disc->d_endian setting.  If the second read succeeds, then
 * the new pads->disc->d_endian setting is retained, otherwise the
 * original pads->disc->d_endian setting is retained.
 */
```

```
/*
 *
 * N.B. The Pendian annotatoin is only able to determine the correct
 * endian choice for a field that has an attached constraint, where
 * the wrong choice of endian setting will always cause the constraint
 * to fail.  (In the above example, if a value < 10 is read with the
 * wrong d_endian setting, the result is a value that is much greater
 * than 10.)
 *
 * For all cases, if the specified number of bytes is available, it is
 * always read.  If the width is not available:
 *    + pd->loc.b/e set to elt/char position of start/end of the
 *       'too small' field
 *    + IO cursor is not advanced
 *    + If P_Test_NotIgnore(*m), pd->errCode set to
 *       P_WIDTH_NOT_AVAILABLE, pd->nerr set to 1,
 *       and an error is reported
 */
```

```
Perror_t Pb_int8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint8 *res_out);

Perror_t Pb_int16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint16 *res_out);

Perror_t Pb_int32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint32 *res_out);

Perror_t Pb_int64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pint64 *res_out);

Perror_t Pb_uint8_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint8 *res_out);

Perror_t Pb_uint16_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint16 *res_out);

Perror_t Pb_uint32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint32 *res_out);

Perror_t Pb_uint64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Puint64 *res_out);
```

```
/* ================================================================================
 * WRITE
 */

ssize_t Pb_int8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint8 *val);

ssize_t Pb_int16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint16 *val);

ssize_t Pb_int32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint32 *val);

ssize_t Pb_int64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pint64 *val);

ssize_t Pb_uint8_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint8 *val);

ssize_t Pb_uint16_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint16 *val);

ssize_t Pb_uint32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint32 *val);

ssize_t Pb_uint64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Puint64 *val);

ssize_t Pb_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent
                    );

ssize_t Pb_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent
                    );
```

```
ssize_t Pb_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint8 *val
                  );

ssize_t Pb_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint16 *val
                  );

ssize_t Pb_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint32 *val
                  );

ssize_t Pb_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint64 *val
                  );

ssize_t Pb_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint8 *val
                  );

ssize_t Pb_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint16 *val
                  );

ssize_t Pb_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *val
                  );

ssize_t Pb_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint64 *val
                  );

ssize_t Pb_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint8 *val, const char *tag, int indent
                  );

ssize_t Pb_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint16 *val, const char *tag, int indent
                  );

ssize_t Pb_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint32 *val, const char *tag, int indent
                  );

ssize_t Pb_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pint64 *val, const char *tag, int indent
                  );

ssize_t Pb_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint8 *val, const char *tag, int indent
                  );

ssize_t Pb_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint16 *val, const char *tag, int indent
                  );

ssize_t Pb_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint32 *val, const char *tag, int indent
                  );

ssize_t Pb_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Puint64 *val, const char *tag, int indent
                  );
```

```
ssize_t Pb_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep
                    );

ssize_t Pb_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep
                    );

ssize_t Pb_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep
                    );

ssize_t Pb_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep
                    );

ssize_t Pb_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep
                    );

ssize_t Pb_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep
                    );

ssize_t Pb_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep
                    );

ssize_t Pb_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep
                    );
```

```
ssize_t Pb_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep
                  );

ssize_t Pb_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep
                  );

ssize_t Pb_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep
                  );

ssize_t Pb_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep
                  );

ssize_t Pb_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep
                  );

ssize_t Pb_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep
                  );

ssize_t Pb_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep
                  );

ssize_t Pb_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep
                  );
```

```
ssize_t Pb_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint8 *rep
                   );

ssize_t Pb_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint16 *rep
                   );

ssize_t Pb_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint32 *rep
                   );

ssize_t Pb_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Pint64 *rep
                   );

ssize_t Pb_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                   Pbase_m *m, Pbase_pd *pd, Puint8 *rep
                   );

ssize_t Pb_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep
                   );

ssize_t Pb_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep
                   );

ssize_t Pb_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                   const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep
                   );
```

B.8 Integers: Serialized binary encoding

```
/* ================================================================================
 * READ
 */

/* ================================================================================
 * SBL, and SBH ENCODINGS OF INTEGERS (VARIABLE NUMBER OF BYTES)
 *
 * These functions parse signed or unsigned SBL (sbl_) or SBH (sbh_)
 * encoded integers with a specified number of bytes.
 *
 * SBL (Serialized Binary, Low-Order Byte First) INTEGER ENCODING
 *   (Psbl_int_read / Psbl_uint_read):
 *
 *   For a K-byte SBL encoding, the first byte on disk is treated
 *   as the low order byte of a K byte value.
 *
 * SBH (Serialized Binary, High-Order Byte First) INTEGER ENCODING
 *   (Psbh_int_read / Psbh_uint_read):
 *
 *   For a K-byte SBH encoding, the first byte on disk is treated
 *   as the high order byte of a K byte value.
 */
```

```
/* For SBL and SBH, each byte is moved to the in-memory target integer
 * unchanged.  Whether the result is treated as a signed or unsigned
 * number depends on the target type.
 *
 * Note that SBL and SBH differ from the common width binary (B)
 * types in 3 ways: (1) SBL and SBH support any number of
 * bytes between 1 and 8, while B only supports 1, 2, 4, and 8; (2)
 * with SBL and SBH you specify the target type independently of the
 * num_bytes; (3) SBL and SBH explicitly state the byte ordering,
 * while B uses the pads->disc->d_endian setting to determine the byte
 * ordering of the data.
 *
 * The legal range of values for num_bytes
 * depends on target type:
 *
 * Type        num_bytes Min/Max values
 * ----------- --------- ----------------------------
 * Pint8       1-1       P_MIN_INT8  / P_MAX_INT8
 * Puint8      1-1       0           / P_MAX_UINT8
 * Pint16      1-2       P_MIN_INT16 / P_MAX_INT16
 * Puint16     1-2       0           / P_MAX_UINT16
 * Pint32      1-4       P_MIN_INT32 / P_MAX_INT32
 * Puint32     1-4       0           / P_MAX_UINT32
 * Pint64      1-8       P_MIN_INT64 / P_MAX_INT64
 * Puint64     1-8       0           / P_MAX_UINT64
 */
```

```
/* If the specified number of bytes is NOT available:
 *    + pd->loc.b/e set to elt/char position of start/end of the 'too small' field
 *    + IO cursor is not advanced
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_WIDTH_NOT_AVAILABLE,
 *         pd->nerr set to 1, and an error is reported
 *
 * Otherwise, the IO cursor is always advanced.  The error cases that
 * can occur even though the IO cursor advances:
 *
 * If num_bytes is not a legal choice for the target type and
 * sign of the value:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_BAD_PARAM,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the specified bytes make up an integer that does not fit in the target type,
 * or if the actual value is not in the min/max range, then:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 */
```

```
Perror_t Psbl_int8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbl_int16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbl_int32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbl_int64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                        Puint32 num_bytes
                        );
```

```
Perror_t Psbl_uint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbl_uint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint16 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbl_uint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbl_uint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint64 *res_out,
                        Puint32 num_bytes
                        );
```

```
Perror_t Psbh_int8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbh_int16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbh_int32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                        Puint32 num_bytes
                        );

Perror_t Psbh_int64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                        Puint32 num_bytes
                        );
```

```
Perror_t Psbh_uint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                    Puint32 num_bytes
                    );

Perror_t Psbh_uint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint16 *res_out,
                    Puint32 num_bytes
                    );

Perror_t Psbh_uint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                    Puint32 num_bytes
                    );

Perror_t Psbh_uint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint64 *res_out,
                    Puint32 num_bytes
                    );




/* =================================================================================
 * WRITE
 */



ssize_t Psbl_int8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    Puint32 num_bytes
                    );




ssize_t Psbl_uint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbl_uint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbl_uint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbl_uint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    Puint32 num_bytes
                    );
```

```
ssize_t Psbl_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbl_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbl_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbl_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbl_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbl_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbl_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, Puint32 num_bytes
                    );

ssize_t Psbl_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, Puint32 num_bytes
                    );

ssize_t Psbl_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, Puint32 num_bytes
                    );

ssize_t Psbl_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, Puint32 num_bytes
                    );

ssize_t Psbl_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, Puint32 num_bytes
                    );

ssize_t Psbl_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, Puint32 num_bytes
                    );

ssize_t Psbl_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint16 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbl_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbl_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint64 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint8 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbl_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbl_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbl_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint16 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint16 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint64 *rep, Puint32 num_bytes
                    );

ssize_t Psbl_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint64 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbl_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint8 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbl_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                    Puint32 num_bytes
                    );
```

```
ssize_t Psbl_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbl_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbl_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbl_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbh_uint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbh_uint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    Puint32 num_bytes
                    );

ssize_t Psbh_uint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbh_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbh_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbh_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbh_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbh_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );

ssize_t Psbh_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, Puint32 num_bytes
                    );

ssize_t Psbh_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, Puint32 num_bytes
                    );

ssize_t Psbh_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, Puint32 num_bytes
                    );

ssize_t Psbh_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, Puint32 num_bytes
                    );

ssize_t Psbh_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, Puint32 num_bytes
                    );

ssize_t Psbh_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, Puint32 num_bytes
                    );

ssize_t Psbh_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint16 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbh_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbh_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint64 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint8 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbh_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbh_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );

ssize_t Psbh_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                    int indent, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint8 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint16 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint16 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint32 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Puint64 *rep, Puint32 num_bytes
                    );

ssize_t Psbh_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                    Pbase_pd *pd, Puint64 *rep, Puint32 num_bytes
                    );
```

```
ssize_t Psbh_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint8 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbh_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                    Puint32 num_bytes
                    );
```

```
ssize_t Psbh_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbh_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbh_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Puint32 num_bytes
                    );

ssize_t Psbh_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    Puint32 num_bytes
                    );
```

B.9 Integers: EBCDIC numeric encoding

```
/* ================================================================================
 * READ
 */

/* ================================================================================
 * EBC ENCODING OF INTEGERS (VARIABLE NUMBER OF DIGITS)
 *
 * These functions parse signed or unsigned EBCDIC numeric (ebc_)
 * encoded integers with a specified number of digits.
 *
 * Each byte on disk encodes one digit (in low 4 bits).  For signed
 * values, the final byte encodes the sign (high 4 bits == 0xD for negative).
 * A signed or unsigned 5 digit value is encoded in 5 bytes.
 *
 * The legal range of values for num_digits
 * depends on target type:
 *
 * Type        num_digits    Min/Max values
 * ----------- ----------    ----------------------------
 * Pint8       1-3           P_MIN_INT8  / P_MAX_INT8
 * Puint8      1-3           0           / P_MAX_UINT8
 * Pint16      1-5           P_MIN_INT16 / P_MAX_INT16
 * Puint16     1-5           0           / P_MAX_UINT16
 * Pint32      1-10          P_MIN_INT32 / P_MAX_INT32
 * Puint32     1-10          0           / P_MAX_UINT32
 * Pint64      1-19          P_MIN_INT64 / P_MAX_INT64
 * Puint64     1-20          0           / P_MAX_UINT64
 *
 * N.B.: num_digits must be odd if the value on disk can be negative.
 */
```

```
/*
 * If the required number of bytes is NOT available:
 *    + pd->loc.b/e set to elt/char position of start/end of the 'too small' field
 *    + IO cursor is not advanced
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_WIDTH_NOT_AVAILABLE,
 *         pd->nerr set to 1, and an error is reported
 *
 * Otherwise, the IO cursor is always advanced.  The error cases that
 * can occur even though the IO cursor advances:
 *
 * If num_bytes is not a legal choice for the target type and
 * sign of the value:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_BAD_PARAM,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the specified digits make up an integer that does not fit in the target type,
 * or if the actual value is not in the min/max range, then:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the specified bytes are not legal EBC integer bytes, then
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_INVALID_EBC_NUM
 *         pd->nerr set to 1, and an error is reported
 */


Perror_t Pebc_int8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_int16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_int32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_int64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_uint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_uint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint16 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_uint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                    Puint32 num_digits
                    );

Perror_t Pebc_uint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint64 *res_out,
                    Puint32 num_digits
                    );


/* =================================================================================
 * WRITE
 */
```

```
ssize_t Pebc_int8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_int16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_int32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_int64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    Puint32 num_digits
                    );
```

```
ssize_t Pebc_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );
```

```
ssize_t Pebc_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, Puint32 num_digits
                    );

ssize_t Pebc_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, Puint32 num_digits
                    );

ssize_t Pebc_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, Puint32 num_digits
                    );

ssize_t Pebc_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, Puint32 num_digits
                    );

ssize_t Pebc_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, Puint32 num_digits
                    );

ssize_t Pebc_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, Puint32 num_digits
                    );

ssize_t Pebc_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, Puint32 num_digits
                    );

ssize_t Pebc_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, Puint32 num_digits
                    );
```

```
ssize_t Pebc_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent,
                    Puint32 num_digits
                    );

ssize_t Pebc_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint16 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pebc_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pebc_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint64 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint8 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pebc_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );
```

```
ssize_t Pebc_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint8 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint8 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint16 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint16 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint32 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint32 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint64 *rep, Puint32 num_digits
                  );

ssize_t Pebc_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pint64 *rep, Puint32 num_digits
                  );
```

```
ssize_t Pebc_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint8 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint8 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint16 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                   Pbase_pd *pd, Puint16 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint32 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                   Pbase_pd *pd, Puint32 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                   Puint64 *rep, Puint32 num_digits
                   );

ssize_t Pebc_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                   int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                   Pbase_pd *pd, Puint64 *rep, Puint32 num_digits
                   );
```

```
ssize_t Pebc_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint8 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Puint32 num_digits
                    );

ssize_t Pebc_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    Puint32 num_digits
                    );
```

B.10 Integers: BCD numeric encoding

```
/* ================================================================================
 * READ
 */

/* ================================================================================
 * BCD ENCODINGS OF INTEGERS (VARIABLE NUMBER OF DIGITS)
 *
 * These functions parse signed or unsigned (bcd_),
 * encoded integers with a specified number of digits.
 *
 * Each byte on disk encodes two digits, 4 bits per digit.  For signed
 * values, a negative number is encoded by having number of digits be
 * odd so that the remaining low 4 bits in the last byte are available
 * for the sign.  (low 4 bits == 0xD for negative).  A signed or
 * unsigned 5 digit value is encoded in 3 bytes, where the unsigned
 * value ignores the final 4 bits and the signed value uses them to
 * get the sign.
 */
```

```
/*
 * The legal range of values for num_digits
 * depends on target type:
 *
 * Type        num_digits    Min/Max values
 * ----------- ----------    ----------------------------
 * Pint8       1-3           P_MIN_INT8  / P_MAX_INT8
 * Puint8      1-3           0           / P_MAX_UINT8
 * Pint16      1-5           P_MIN_INT16 / P_MAX_INT16
 * Puint16     1-5           0           / P_MAX_UINT16
 * Pint32      1-10/11**     P_MIN_INT32 / P_MAX_INT32
 * Puint32     1-10          0           / P_MAX_UINT32
 * Pint64      1-19          P_MIN_INT64 / P_MAX_INT64
 * Puint64     1-20          0           / P_MAX_UINT64
 *
 * N.B.: num_digits must be odd if the value on disk can be negative.
 *
 * ** For Pbcd_int32_read only, even though the min and max int32 have
 * 10 digits, we allow num_digits == 11 due to the fact that 11 is
 * required for a 10 digit negative value (an actual 11 digit number
 * would cause a range error, so the leading digit must be 0).
 */



/*
 * If the required number of bytes is NOT available:
 *    + pd->loc.b/e set to elt/char position of start/end of the
 *       'too small' field
 *    + IO cursor is not advanced
 *    + if P_Test_NotIgnore(*m), pd->errCode set to
 *       P_WIDTH_NOT_AVAILABLE, pd->nerr set to 1,
 *       and an error is reported
 *
 * Otherwise, the IO cursor is always advanced.  The error cases that
 * can occur even though the IO cursor advances:
 *
 * If num_digits is not a legal choice for the target type and
 * sign of the value:
 *    + pd->loc.b/e set to elt/char position at the start/end
 *       of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_BAD_PARAM,
 *         pd->nerr set to 1, and an error is reported
 */



/*
 * If the specified bytes make up an integer that does not fit in the
 * target type, or if the actual value is not in the min/max range,
 * then:
 *    + pd->loc.b/e set to elt/char position at the start/end
 *      of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the specified bytes are not legal BCD integer bytes, then
 *    + pd->loc.b/e set to elt/char position at the start/end
 *       of the field
 *    + if P_Test_NotIgnore(*m), pd->errCode set to
 *       P_INVALID_BCD_NUM, pd->nerr set to 1,
 *       and an error is reported
 */
```

```
Perror_t Pbcd_int8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint8 *res_out,
                        Puint32 num_digits
                        );

Perror_t Pbcd_int16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint16 *res_out,
                         Puint32 num_digits
                         );

Perror_t Pbcd_int32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint32 *res_out,
                         Puint32 num_digits
                         );

Perror_t Pbcd_int64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Pint64 *res_out,
                         Puint32 num_digits
                         );

Perror_t Pbcd_uint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint8 *res_out,
                         Puint32 num_digits
                         );

Perror_t Pbcd_uint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint16 *res_out,
                          Puint32 num_digits
                          );

Perror_t Pbcd_uint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint32 *res_out,
                          Puint32 num_digits
                          );

Perror_t Pbcd_uint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd, Puint64 *res_out,
                          Puint32 num_digits
                          );
```

```
ssize_t Pbcd_int8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_int16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_int32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_int64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    Puint32 num_digits
                    );
```

```
ssize_t Pbcd_int8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint8 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_int16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint16 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_int32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint32 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_int64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pint64 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint8 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint16 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint32 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Puint64 *val,
                    const char *tag, int indent, Puint32 num_digits
                    );
```

```
ssize_t Pbcd_int8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, Puint32 num_digits
                    );

ssize_t Pbcd_int16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint16 *val, Puint32 num_digits
                    );

ssize_t Pbcd_int32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint32 *val, Puint32 num_digits
                    );

ssize_t Pbcd_int64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint64 *val, Puint32 num_digits
                    );

ssize_t Pbcd_uint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint8 *val, Puint32 num_digits
                    );

ssize_t Pbcd_uint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint16 *val, Puint32 num_digits
                    );

ssize_t Pbcd_uint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint32 *val, Puint32 num_digits
                    );

ssize_t Pbcd_uint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Puint64 *val, Puint32 num_digits
                    );
```

```
ssize_t Pbcd_int8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pint8 *val, const char *tag, int indent,
                    Puint32 num_digits
                    );

ssize_t Pbcd_int16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint16 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pbcd_int32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint32 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pbcd_int64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pint64 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint8 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint16 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint32 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );

ssize_t Pbcd_uint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Puint64 *val, const char *tag,
                    int indent, Puint32 num_digits
                    );
```

```
ssize_t Pbcd_int8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint8 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint16 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint32 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, Puint32 num_digits
                    );

ssize_t Pbcd_int64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                    Pint64 *rep, Puint32 num_digits
                    );
```

```
ssize_t Pbcd_uint8_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint8_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint8 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint16_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint16 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint16_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint16 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint32 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint32 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Puint64 *rep, Puint32 num_digits
                  );

ssize_t Pbcd_uint64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, int *requested_out, const char *delims, Pbase_m *m,
                  Pbase_pd *pd, Puint64 *rep, Puint32 num_digits
                  );
```

```
ssize_t Pbcd_int8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pint8 *rep,
                  Puint32 num_digits
                  );

ssize_t Pbcd_int16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pint16 *rep,
                  Puint32 num_digits
                  );

ssize_t Pbcd_int32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pint32 *rep,
                  Puint32 num_digits
                  );

ssize_t Pbcd_int64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pint64 *rep,
                  Puint32 num_digits
                  );
```

```
ssize_t Pbcd_uint8_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint8 *rep,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint16_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint16 *rep,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint32 *rep,
                    Puint32 num_digits
                    );

ssize_t Pbcd_uint64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                    const char *delims, Pbase_m *m, Pbase_pd *pd, Puint64 *rep,
                    Puint32 num_digits
                    );
```

B.11 Fixed Point: EBCDIC numeric encoding

```
/* ================================================================================
 * READ
 */

/* ================================================================================
 * FIXED POINT READ FUNCTIONS
 * FOR EBC (ebc_) ENCODING
 *
 * An in-memory fpoint or ufpoint number is a signed or unsigned
 * fixed-point rational number with a numerator and denominator that
 * both have the same size.  For signed fpoint types, the numerator
 * carries the sign, while the denominator is always unsigned.  For
 * example, type Pfpoint16 has a signed Pint16 numerator and an
 * unsigned Puint16 denominator.
 *
 * For the EBC fpoint read functions, num_digits is the
 * number of digits used to encode the numerator (on disk). The number
 * of bytes implied by num_digits is the same as specified for the
 * EBC integer read functions.
 *
 * For all fpoint types, d_exp determines the denominator value,
 * which is implicitly 10^d_exp and is not encoded on disk.
 * The legal range of values for d_exp depends on the type:
 *
 * Type                     d_exp     Max denominator (min is 1)
 * -----------------------  --------  --------------------------
 * Pfpoint8  /  ufpoint8  0-2                             100
 * Pfpoint16 / ufpoint16  0-4                          10,000
 * Pfpoint32 / ufpoint32  0-9                   1,000,000,000
 * Pfpoint64 / ufpoint64  0-19     10,000,000,000,000,000,000
 */
```

```
/*
 * The legal range of values for num_digits
 * depends on target type, and is the same as specified for the
 * EBC integer read functions.
 *
 * If the specified number of bytes are NOT available:
 *    + pd->loc.b/e set to elt/char position of start/end of the 'too small' field
 *    + IO cursor not advanced
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_WIDTH_NOT_AVAILABLE,
 *         pd->nerr set to 1, and an error is reported
 *
 * Otherwise, the IO cursor is always advanced.  The error cases that
 * can occur even though the IO cursor advances:
 *
 * If num_digits or d_exp is not in a legal choice for the target type
 * and sign of the value:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_BAD_PARAM,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the actual numerator is not in the min/max numerator range, then:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the specified bytes are not legal EBC integer bytes, then
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_INVALID_EBC_NUM
 *         pd->nerr set to 1, and an error is reported
 *
 */


Perror_t Pebc_fpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint8 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_fpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint16 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_fpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint32 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_fpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint64 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_ufpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint8 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_ufpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint16 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_ufpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint32 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pebc_ufpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint64 *res_out, Puint32 num_digits, Puint32 d_exp
                   );
```

```
ssize_t Pebc_fpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint16 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint32 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint64 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint8 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint16 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint32 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint64 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );
```

```
ssize_t Pebc_fpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                   const char *tag, int indent, Puint32 num_digits, Puint32 d_exp
                   );

ssize_t Pebc_fpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint16 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pebc_fpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint32 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pebc_fpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint64 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pebc_ufpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint8 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pebc_ufpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint16 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pebc_ufpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint32 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pebc_ufpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint64 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );
```

```
ssize_t Pebc_fpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint8 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint16 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint32 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint64 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint8 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint16 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint32 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint64 *val, Puint32 num_digits, Puint32 d_exp
                    );
```

```
ssize_t Pebc_fpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint8 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint16 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint32 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_fpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint64 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint8 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint16 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint32 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pebc_ufpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint64 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );
```

B.12 Fixed Point: BCD numeric encoding

```
/* ============================================================================
 * READ
 */

/* ============================================================================
 * FIXED POINT READ FUNCTIONS
 * BCD (bcd_) ENCODINGS
 *
 * An in-memory fpoint or ufpoint number is a signed or unsigned
 * fixed-point rational number with a numerator and denominator that
 * both have the same size.  For signed fpoint types, the numerator
 * carries the sign, while the denominator is always unsigned.  For
 * example, type Pfpoint16 has a signed Pint16 numerator and an
 * unsigned Puint16 denominator.
 *
 * For the BCD fpoint read functions, num_digits is the number of
 * digits used to encode the numerator (on disk). The number of bytes
 * implied by num_digits is the same as specified for the BCD integer
 * read functions.
 *
 * For all fpoint types, d_exp determines the denominator value, which
 * is implicitly 10^d_exp and is not encoded on disk.  The legal range
 * of values for d_exp depends on the type:
 *
 * Type                     d_exp     Max denominator (min is 1)
 * -----------------------  --------  --------------------------
 * Pfpoint8  /  ufpoint8  0-2                              100
 * Pfpoint16 / ufpoint16  0-4                           10,000
 * Pfpoint32 / ufpoint32  0-9                    1,000,000,000
 * Pfpoint64 / ufpoint64  0-19     10,000,000,000,000,000,000
 *
 * The legal range of values for num_digits depends on target type,
 * and is the same as specified for the BCD integer read functions.
 */
```

```
/*
 * For all cases, if the specified number of bytes are NOT available:
 *    + pd->loc.b/e set to elt/char position of start/end of the
 *      'too small' field
 *    + IO cursor not advanced
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to
 *       P_WIDTH_NOT_AVAILABLE, pd->nerr set to 1,
 *       and an error is reported
 *
 * Otherwise, the IO cursor is always advanced.  The error cases that
 * can occur even though the IO cursor advances:
 *
 * If num_digits or d_exp is not in a legal choice for the target type
 * and sign of the value:
 *    + pd->loc.b/e set to elt/char position at the start/end of the
 *      field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_BAD_PARAM,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the actual numerator is not in the min/max numerator range,
 * then:
 *    + pd->loc.b/e set to elt/char position at the start/end
 *       of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the specified bytes are not legal BCD integer bytes, then
 *    + pd->loc.b/e set to elt/char position at the start/end
 *       of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to
 *       P_INVALID_BCD_NUM, pd->nerr set to 1,
 *       and an error is reported
 *
 */
```

```
Perror_t Pbcd_fpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint8 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_fpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint16 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_fpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint32 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_fpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint64 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_ufpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint8 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_ufpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint16 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_ufpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint32 *res_out, Puint32 num_digits, Puint32 d_exp
                   );

Perror_t Pbcd_ufpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint64 *res_out, Puint32 num_digits, Puint32 d_exp
                   );
```

```
ssize_t Pbcd_fpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint16 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint32 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint64 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint8 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint16 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint32 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint64 *val,
                    Puint32 num_digits, Puint32 d_exp
                    );
```

```
ssize_t Pbcd_fpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                   const char *tag, int indent, Puint32 num_digits, Puint32 d_exp
                   );

ssize_t Pbcd_fpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint16 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pbcd_fpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint32 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pbcd_fpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint64 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pbcd_ufpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint8 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pbcd_ufpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint16 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pbcd_ufpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint32 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );

ssize_t Pbcd_ufpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint64 *val, const char *tag, int indent, Puint32 num_digits,
                   Puint32 d_exp
                   );
```

```
ssize_t Pbcd_fpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint8 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint16 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint32 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint64 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint8 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint16 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint32 *val, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint64 *val, Puint32 num_digits, Puint32 d_exp
                    );
```

```
ssize_t Pbcd_fpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint8 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint16 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint32 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_fpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint64 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint8 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint16 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint32 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );

ssize_t Pbcd_ufpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint64 *val, const char *tag,
                    int indent, Puint32 num_digits, Puint32 d_exp
                    );
```

B.13 Fixed Point: Serialized binary numeric encoding

```
/* ============================================================================
 * READ
 */

/* ============================================================================
 * FIXED POINT READ FUNCTIONS
 * SBL (sbl_), and SBH (sbh_) ENCODINGS
 *
 * An in-memory fpoint or ufpoint number is a signed or unsigned
 * fixed-point rational number with a numerator and denominator that
 * both have the same size.  For signed fpoint types, the numerator
 * carries the sign, while the denominator is always unsigned.  For
 * example, type Pfpoint16 has a signed Pint16 numerator and an
 * unsigned Puint16 denominator.
 *
 * For the SBL and SBH fpoint read functions, num_bytes is the number
 * of bytes on disk used to encode the numerator, the encoding being
 * the same as for the SBL and SBH integer read functions,
 * respectively.
 */



/* For all fpoint types, d_exp determines the denominator value,
 * which is implicitly 10^d_exp and is not encoded on disk.
 * The legal range of values for d_exp depends on the type:
 *
 * Type                     d_exp     Max denominator (min is 1)
 * -----------------------  --------  --------------------------
 * Pfpoint8  /  ufpoint8  0-2                              100
 * Pfpoint16 / ufpoint16  0-4                           10,000
 * Pfpoint32 / ufpoint32  0-9                    1,000,000,000
 * Pfpoint64 / ufpoint64  0-19     10,000,000,000,000,000,000
 *
 * The legal range of values for num_bytes
 * depends on target type, and is the same as specified for the
 * SBL/SBH integer read functions.
 */



/* If the specified number of bytes are NOT available:
 *    + pd->loc.b/e set to elt/char position of start/end of the 'too small' field
 *    + IO cursor not advanced
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_WIDTH_NOT_AVAILABLE,
 *         pd->nerr set to 1, and an error is reported
 *
 * Otherwise, the IO cursor is always advanced.  The error cases that
 * can occur even though the IO cursor advances:
 *
 * If num_bytes or d_exp is not in a legal choice for the target type
 * and sign of the value:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_BAD_PARAM,
 *         pd->nerr set to 1, and an error is reported
 *
 * If the actual numerator is not in the min/max numerator range, then:
 *    + pd->loc.b/e set to elt/char position at the start/end of the field
 *    + if PD_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 */
```

```
Perror_t Psbl_fpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint8 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbl_fpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint16 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbl_fpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint32 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbl_fpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint64 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );
```

```
Perror_t Psbl_ufpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint8 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbl_ufpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint16 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbl_ufpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint32 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbl_ufpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pufpoint64 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );
```

```
Perror_t Psbh_fpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint8 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbh_fpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint16 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbh_fpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint32 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );

Perror_t Psbh_fpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                   Pfpoint64 *res_out, Puint32 num_bytes, Puint32 d_exp
                   );
```

```
Perror_t Psbh_ufpoint8_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Pufpoint8 *res_out, Puint32 num_bytes, Puint32 d_exp
                    );

Perror_t Psbh_ufpoint16_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Pufpoint16 *res_out, Puint32 num_bytes, Puint32 d_exp
                    );

Perror_t Psbh_ufpoint32_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Pufpoint32 *res_out, Puint32 num_bytes, Puint32 d_exp
                    );

Perror_t Psbh_ufpoint64_read(P_t *pads, const Pbase_m *m, Pbase_pd *pd,
                    Pufpoint64 *res_out, Puint32 num_bytes, Puint32 d_exp
                    );
```

```
/* =================================================================================
 * WRITE
 */
```

```
ssize_t Psbl_fpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbl_fpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint16 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbl_fpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint32 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbl_fpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint64 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbl_ufpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint8 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbl_ufpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint16 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbl_ufpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint32 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbl_ufpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint64 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbl_fpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                   const char *tag, int indent, Puint32 num_bytes, Puint32 d_exp
                   );

ssize_t Psbl_fpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint16 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );

ssize_t Psbl_fpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint32 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );

ssize_t Psbl_fpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pfpoint64 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );




ssize_t Psbl_ufpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint8 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );

ssize_t Psbl_ufpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint16 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );

ssize_t Psbl_ufpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint32 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );

ssize_t Psbl_ufpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                   Pufpoint64 *val, const char *tag, int indent, Puint32 num_bytes,
                   Puint32 d_exp
                   );




ssize_t Psbl_fpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pfpoint8 *val, Puint32 num_bytes, Puint32 d_exp
                   );

ssize_t Psbl_fpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pfpoint16 *val, Puint32 num_bytes, Puint32 d_exp
                   );

ssize_t Psbl_fpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pfpoint32 *val, Puint32 num_bytes, Puint32 d_exp
                   );

ssize_t Psbl_fpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                   Pbase_pd *pd, Pfpoint64 *val, Puint32 num_bytes, Puint32 d_exp
                   );
```

```
ssize_t Psbl_ufpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pufpoint8 *val, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_ufpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pufpoint16 *val, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_ufpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pufpoint32 *val, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_ufpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pufpoint64 *val, Puint32 num_bytes, Puint32 d_exp
                  );
```

```
ssize_t Psbl_fpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfpoint8 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_fpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfpoint16 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_fpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfpoint32 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_fpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfpoint64 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );
```

```
ssize_t Psbl_ufpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pufpoint8 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_ufpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pufpoint16 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_ufpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pufpoint32 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );

ssize_t Psbl_ufpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pufpoint64 *val, const char *tag,
                  int indent, Puint32 num_bytes, Puint32 d_exp
                  );
```

```
ssize_t Psbh_fpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint16 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint32 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint64 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbh_ufpoint8_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint8 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint16_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint16 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint32 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pufpoint64 *val,
                    Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbh_fpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfpoint8 *val,
                    const char *tag, int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pfpoint16 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );

ssize_t Psbh_fpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pfpoint32 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );

ssize_t Psbh_fpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pfpoint64 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );
```

```
ssize_t Psbh_ufpoint8_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pufpoint8 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );

ssize_t Psbh_ufpoint16_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pufpoint16 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );

ssize_t Psbh_ufpoint32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pufpoint32 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );

ssize_t Psbh_ufpoint64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd,
                    Pufpoint64 *val, const char *tag, int indent, Puint32 num_bytes,
                    Puint32 d_exp
                    );
```

```
ssize_t Psbh_fpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint8 *val, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint16 *val, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint32 *val, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pfpoint64 *val, Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbh_ufpoint8_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint8 *val, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint16_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint16 *val, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint32 *val, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                    Pbase_pd *pd, Pufpoint64 *val, Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbh_fpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint8 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint16 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint32 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_fpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pfpoint64 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );
```

```
ssize_t Psbh_ufpoint8_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint8 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint16_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint16 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint32 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );

ssize_t Psbh_ufpoint64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                    int *buf_full, Pbase_pd *pd, Pufpoint64 *val, const char *tag,
                    int indent, Puint32 num_bytes, Puint32 d_exp
                    );
```

B.14 Floats: Character encodings

```
/* ============================================================================
 * READ
 */

/* ============================================================================
 * CHARACTER-BASED FLOAT READ FUNCTIONS
 *
 * DEFAULT                        ASCII                          EBCDIC
 * -----------------------------  -----------------------------  -----------------------------
 * Pfloat32_read                  Pa_float32_read                Pe_float32_read
 * Pfloat64_read                  Pa_float64_read                Pe_float64_read
 *
 * These types describe ASCII or EBCDIC character-based encodings of
 * floating point numbers.  The input representation must have this
 * form:
 *
 *   [+|-]DIGITS[.][DIGITS][(e|E)[+|-]DIGITS]
 *
 * Where DIGITS is a sequence of one or more
 * digit characters, (e|E) indicates either a lower- or
 * upper-case letter 'E', and elements in square brackets are
 * optional.  Note that there must be at least one digit before the
 * decimal point.
 *
 * If the input has a valid sequence of input characters that make up
 * a float, then the float is converted to a Pfloat32 or
 * Pfloat64, according to the type.  For example, if you specify
 * a Pa_float32 then ASCII characters making up a float will be read
 * from the input and converted to an in-memory Pfloat32.
 *
 * RETURN VALUE: Perror_t
 */
```

```
/*
 * Upon success, P_OK returned:
 *   + the IO cursor is advanced to just beyond the last digit
 *   + if P_Test_NotIngore(*m), the out param is assigned a value
 *
 * P_ERR is returned on error.
 * Cursor advancement/err settings for different error cases:
 *
 * (1) If IO cursor is at EOF
 *     + pd->loc.b/e set to EOF 'location'
 *     + IO cursor remains at EOF
 *     + if P_Test_NotIgnore(*), pd->errCode set to P_AT_EOF,
 *         pd->nerr set to 1, and an error is reported
 * (2a) There is leading white space and not (pads->disc->flags & P_WSPACE_OK)
 * (2b) The target is unsigned and the first char is a -
 * (2c) The first character is not a +, -, or in [0-9]
 * (2d) First character is allowable + or -, following by a char that is not a digit
 * For the above 4 cases:
 *     + pd->loc.b/e set to the IO cursor position.
 *     + IO cursor is not advanced
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_INVALID_A_NUM/P_INVALID_E_NUM,
 *         pd->nerr set to 1, and an error is reported
 * (3) A valid ASCII/EBCDIC float is found, but it describes
 *     a float that does not fit in the specified target type
 *     + pd->loc.b/e set to elt/char position of start and end of the float
 *     + IO cursor is advanced just beyond the last digit
 *     + if P_Test_NotIgnore(*m), pd->errCode set to P_RANGE,
 *         pd->nerr set to 1, and an error is reported
 */

#if P_CONFIG_READ_FUNCTIONS > 0

#if P_CONFIG_A_FLOAT > 0


Perror_t Pa_float32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pfloat32 *res_out);

Perror_t Pa_float64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pfloat64 *res_out);







Perror_t Pe_float32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pfloat32 *res_out);

Perror_t Pe_float64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pfloat64 *res_out);







Perror_t Pfloat32_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pfloat32 *res_out);

Perror_t Pfloat64_read(P_t *pads,const Pbase_m *m,Pbase_pd *pd,Pfloat64 *res_out);
```

```
ssize_t Pa_float32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat32 *val
                  );

ssize_t Pa_float64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat64 *val
                  );

ssize_t Pa_float32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat32 *val
                  );

ssize_t Pa_float64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat64 *val
                  );

ssize_t Pa_float32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfloat32 *val, const char *tag,
                  int indent
                  );

ssize_t Pa_float64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfloat64 *val, const char *tag,
                  int indent
                  );

ssize_t Pa_float32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat32 *val,
                  const char *tag, int indent
                  );

ssize_t Pa_float64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat64 *val,
                  const char *tag, int indent
                  );

ssize_t Pa_float32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat32 *rep
                  );

ssize_t Pa_float32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat32 *rep
                  );




ssize_t Pa_float64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat64 *rep
                  );

ssize_t Pa_float64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat64 *rep
                  );

ssize_t Pa_float32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pfloat32 *rep
                  );

ssize_t Pa_float64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pfloat64 *rep
                  );
```

```
ssize_t Pe_float32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat32 *val
                  );

ssize_t Pe_float64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat64 *val
                  );

ssize_t Pe_float32_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat32 *val
                  );

ssize_t Pe_float64_write2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat64 *val
                  );

ssize_t Pe_float32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfloat32 *val, const char *tag,
                  int indent
                  );

ssize_t Pe_float64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len,
                  int *buf_full, Pbase_pd *pd, Pfloat64 *val, const char *tag,
                  int indent
                  );

ssize_t Pe_float32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat32 *val,
                  const char *tag, int indent
                  );

ssize_t Pe_float64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat64 *val,
                  const char *tag, int indent
                  );

ssize_t Pe_float32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat32 *rep
                  );

ssize_t Pe_float32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat32 *rep
                  );




ssize_t Pe_float64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat64 *rep
                  );

ssize_t Pe_float64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat64 *rep
                  );

ssize_t Pe_float32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pfloat32 *rep
                  );

ssize_t Pe_float64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out,
                  const char *delims, Pbase_m *m, Pbase_pd *pd, Pfloat64 *rep
                  );
```

```
ssize_t Pfloat32_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pfloat32 *val);


ssize_t Pfloat64_write2io(P_t *pads,Sfio_t *io,Pbase_pd *pd,Pfloat64 *val);


ssize_t Pfloat32_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat32 *val
                  );

ssize_t Pfloat64_write2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat64 *val
                  );

ssize_t Pfloat32_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat32 *val,
                  const char *tag, int indent
                  );

ssize_t Pfloat64_write_xml_2io(P_t *pads, Sfio_t *io, Pbase_pd *pd, Pfloat64 *val,
                  const char *tag, int indent
                  );

ssize_t Pfloat32_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat32 *val, const char *tag, int indent
                  );

ssize_t Pfloat64_write_xml_2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  Pbase_pd *pd, Pfloat64 *val, const char *tag, int indent
                  );




ssize_t Pfloat32_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat32 *rep
                  );

ssize_t Pfloat32_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat32 *rep
                  );

ssize_t Pfloat64_fmt2buf(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat64 *rep
                  );

ssize_t Pfloat64_fmt2buf_final(P_t *pads, Pbyte *buf, size_t buf_len, int *buf_full,
                  int *requested_out, const char *delims, Pbase_m *m, Pbase_pd *pd,
                  Pfloat64 *rep
                  );

ssize_t Pfloat32_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                  Pbase_m *m, Pbase_pd *pd, Pfloat32 *rep
                  );

ssize_t Pfloat64_fmt2io(P_t *pads, Sfio_t *io, int *requested_out, const char *delims,
                  Pbase_m *m, Pbase_pd *pd, Pfloat64 *rep
                  );
```