Matchmaker:
The Accent Remote Call Procedure Language

August 1984

PERQ Systems Corporation
2600 Liberty Avenue
P.O. Box 2600
Pittsburgh, PA 15230
(412) 355-0900

Accent and many of its subsystems and support programs were originally developed by the CMU Computer Science Department as part of its Spice Project.

The system described in this document is based upon the Matchmaker program by Michael B. Jones of CMU and Joseph Ginder, Ellen Colwell, and Edward Pervin of PERQ Systems.

This document is adapted from the paper by Michael B. Jones, *Matchmaker: A Remote Procedure Call Generator*, Carnegie-Mellon University, Pittsburgh, PA 1983, and from the paper by Michael B. Jones, *Matchmaker: A Language for Remote Procedure Calls*, Carnegie-Mellon University, Pittsburgh, PA 1984.

The information in this document is subject to change without notice and should not be construed as a commitment by PERQ Systems Corporation. The company assumes no responsibility for any errors that may appear in this document. PERQ Systems Corporation will make every effort to keep customers apprised of all documentation changes as quickly as possible.

1. Introduction

1.1. What is Matchmaker?

Matchmaker provides a mechanism for declaratively defining a procedural interface for message-based communication between processes. This type of procedural interface to message-based communication is usually called a remote procedure call. Remote procedure calls in Accent can involve processes written in different languages; Matchmaker supports this capability. Once an interface has been declared, Matchmaker may be used to generate procedures for sending and receiving messages between processes written in any of the supported languages. Matchmaker does all the work of appropriately packing the procedure arguments into messages and extracting incoming procedure arguments from message fields.

Matchmaker allows a programmer to write a server process and declare the types of all data to be exchanged between the server and its client processes. Based on those types, procedural interfaces are declared for sending data between processes in messages. A client process would call a Matchmaker-generated procedure to make use of a server facility. This procedure would pack the procedural parameters into a message and send the message to the server process. The server process would receive the message, provide its service, and pack return data into a message to be sent back to the client process. The client process would return from the original procedure call with the data sent back from the server. Remote procedure call interfaces simplify and increase the reliability of writing message based code.

1.2. Example

In PERQ Pascal, to send a message to a server containing a 32-bit integer and a port, a hand coded program like this is necessary:

```
Function SendIt(ServPort : Port; I : Long;
     P : Port ) : GeneralReturn;
         type
            MyMessage = record
                head      : Msg;
                IPCNam2   : TypeType;
                Arg2      : Long;
                IPCNam3   : TypeType;
                Arg3      : Port;
            end;

         var
                MyMsg     : MyMessage;

  begin
    with MyMsg.head do
      begin
         SimpleMsg := false;
         MsgSize := WordSize(MyMsg)*2;
         MsgType := NORMALMSG;
         RemotePort := ServPort;
         LocalPort := ReplyPort;
         ID :=    10000;
      end;
    with MyMsg do
      begin
         IPCNam2.LongInteger := #2000220002;
         Arg2 := I;
         IPCNam3.LongInteger := #2000220006;
         Arg3 := P;
      end;
    SendIt := Send(MyMsg.head,0,WAIT);
  end;
```

An equivalent piece of code could be generated with these Matchmaker declarations:

```
Interface foo = 10000;

Message SendIt( : port; I : Long; P : Port) : GR_Value;
```

2. Overview

This chapter describes what the Matchmaker language can do and how it works.

2.1. What Does It Do?

Matchmaker, a specialized compiler, accepts its own language and produces code for multiple target "machines". The target "machines" for Matchmaker code generators are standard programming languages: C, Lisp, Pascal.

The result of a compilation is source code for each target language. This code implements the same message interface to the declared service, independent of the language with which it will be used.

2.2. A Declarative Language for Data Structures

Matchmaker defines a set of data type primitives and constructors for declaring types and values and a set of rules for representing these declared types. For any Matchmaker type, the Matchmaker code generator for each language can generate a native type declaration for the desired representation.

Any data types that are to be used by more than one language should be declared in the Matchmaker language. Then Matchmaker can be used to generate the appropriate types for each language.

2.3. A Declarative Language for Remote Procedure Calls

Matchmaker simplifies message passing by providing a set of message-based procedural linkages between processes. A number of classes of remote procedures can be declared. There are enough to cover almost all interactions between cooperating processes.

2.4. Goals

The main goals of Matchmaker are:

1. to provide a self-contained language for specifying message level interfaces between cooperating Accent processes;
2. to provide a language rich enough to express any data structure that can be both efficiently represented in a message and reasonably represented in all target languages; and
3. to generate efficient code.

3. General Language Structure

3.1. Lexical Structure

Identifiers — Identifiers are represented as a string of letters, digits, and underscores. The first character may not be a digit. Case of identifiers is preserved as in the declaration, but identifiers are matched with case folded.

Integers — Decimal integers are represented as a string of decimal digits. Octal integers are represented as a '#' character followed by a string of octal digits.

Strings — Character strings are enclosed in double quotes (""). The literal double quote is represented by two of them. Strings cannot span lines.

Characters — Character constants are surrounded by single quotes('). The literal single quote is represented by a string of four single quotes.

Comments — Comments may be introduced at any lexical break with the exclamation point character (!) and continue to the end of the line.

3.2. Kinds of Matchmaker Specifications

There are two kinds of Matchmaker specification files: Types and Interface.

3.2.1. Types specifications

Types specifications contain data structure declarations, but no code declarations:

Types <SpecificationName>;

where <SpecificationName> is an identifier for the specification. A Types specification must end with the keywords End Types.

3.2.2. Interface specifications

Interface specifications contain data structure declarations and code (call) declarations:

Interface <SpecificationName> = <MsgIDbase> ;

where:

- <SpecificationName> is an identifier for the specification.
- <MsgIDbase> is an expression that names the message id that is to be allocated to the first call declared.

Interface specifications must end with the keywords End Interface.

3.3. Order of Declarations

All Matchmaker source files (.mm) must have declarations in this order:

1. Declare Interfaces or Types: Give name of specification, and, for Interface declaration, message id origin. *Required.*
2. Declare Options: Give values for various options. *Optional.*
3. Declare Data: Can be Constant, Type, or Use declarations. *Optional.*
4. Declare Calls: Used for declaring message-based calls. *Required* for Interface specifications; *not allowed* for Types specifications.
5. End Interface or End Types: End Interface for Interface specifications, or End Types for Types specifications. *Required.*

3.4. Expressions

Typed constant values are used in many different contexts throughout the Matchmaker language. Anywhere a literal constant (such as #400, "Carp", or 'a') can be used, a compile-time constant expression can also be used. Since all Matchmaker expressions are evaluated at the time the specification is compiled, compile-time constant expressions will be referred to as "expressions".

3.4.1. Primitive expressions

The primitive building blocks for Matchmaker expressions are:

Literals	Any integer, string, or character literal
Boolean Literals	The pre-declared identifiers "True" or "False"
Constant Names	The identifier for any value declared in a "Constant" declaration
Enumeration Names	The identifier for any element of an enumerated type

3.4.2. Expression operators

The operators for combining expressions, in decreasing order of precedence are:

(...)	Specify evaluation order
* / Mod	Multiplication, Division, Modulus
+ -	Addition, Subtraction (Either may be unary)
= <>	Equal, Not equal
> >= <= <	Order operators
Not	Boolean negation
And	Conjunction
Or	Disjunction

4. Data Declarations

This chapter will cover the means of declaring typed values and data types in Matchmaker specifications.

4.1. Constant Declarations

Symbolic constant values may be declared to Matchmaker using the Constant keyword, followed by a list of declarations of the form:

Name = Expression ;

4.2. Built-in Constants

The two boolean values True and False are pre-declared in every Matchmaker specification. Their declarations are equivalent to the declarations:

Constant
  True is the same as 0 = 0;
  False is the same as 1 = 0;

4.3. Type Declarations

Named types may be declared to Matchmaker using the Type keyword, followed by a list of declarations of the form:

Name = Type Specification ;

Type Specification may either be a built-in data type, a previously declared type name, or a new type definition. For more information on new data type definition refer to the next chapter.

4.3.1. Examples

Some examples of Type declarations are:

```
Type
    Integer     = Short;
    Bit23       = unsigned[23];

    Complex     =
      record
        Re      : Real;
        Im      : Real;
      end record;

    NewBoolean  = (No, Yes);

Type
    File_Data   = array [*] of Byte;
```

NOTE: * is the token for a variable-sized array.

4.3.2. Built-in types

Matchmaker provides several types and classes of types which are pre-declared for each specification. They are:

Boolean	A one bit logical quantity.
Character	An eight bit character type.
Signed[n]	An n bit signed integer. n defaults to 16 if the [n] is not specified.
Unsigned[n]	An n bit unsigned integer. n defaults to 16 if the [n] is not specified.
n .. m	A subrange type of the integers n to m inclusive.
PERQ_String[n]	An n byte string prefixed by a length byte and padded to a 16-bit boundary. n defaults to 80 if [n] is omitted.
Port, Port_Send, Port_Receive, Port_Ownership, Port_All	Message communication ports, with associated port rights. Port and Port_Send are synonyms.
Real	A 32 bit IEEE floating point numeric type.
Byte	A pre-declared 8 bit unsigned integer type.
Short	A pre-declared 16 bit signed integer type.
Long	A pre-declared 32 bit signed integer type.

4.4. Use Declarations

The Use declaration allows one Matchmaker specification to use data structure declarations from another. Constant, Type, and Use declarations can appear in any order with respect to one another.

4.4.1. Syntax

The syntax for the Use declaration is the keyword Use followed by specifications of the form:

```
InterfaceName from "FileName" ;
```

where InterfaceName is the declared interface name in the specification to be used by the current specification, and FileName is the filename for that specification without the ".mm" extension. (Language-specific declaration file names will also be derived from FileName).

The "use" chain is recursively expanded. In other words, if an interface A uses declarations from interface B, and B uses interface C, then A may also use declarations from C.

4.4.2. Examples

Some examples of Use declarations are:

```
Use
    Foo from "Foo";
    Shoes from "Shoes";
Use
    AccInt from "Accent";
```

5. Defining New Data Types

Matchmaker allows construction of new data types from existing ones. Matchmaker is a language for specifying message-based interfaces between processes, therefore only those types that can be efficiently sent in messages are supported.

5.1. Record Types

A Record type is a group of one or more "fields" made up of an identifier and an associated data type.

5.1.1. Syntax

A Record type specification contains the keyword Record and one or more fields of the form:

FieldName : FieldType ;

followed by the keywords End Record. Each FieldName must be a valid identifier, and each FieldType must be a declared type name or built-in type.

5.1.2. Packed record types

"Packed" record types are like ordinary record types, except that attempts are made to "pack" several small fields together where possible, instead of always aligning them on 16-bit boundaries. The syntax for packed record types is the same as that for ordinary record types, except that the type keyword is "Packed Record."

5.1.3. Examples

Some examples of Record specifications are:

```
Type
    Couple =
      record
        Male    : Person;
        Female  : Person;
      end record;

    Time_Rec =
      packed record
        Hours   : 0 .. 23;
        Minutes : 0 .. 59;
        Seconds : 0 .. 59;
      end record;
```

5.1.4. Restrictions

Due to message passing restrictions, record field types must not be port types, pointer types, variable-sized array types, or union types.

5.2. Array Types

An Array type is a collection of some number of elements, all of the same data type. The number of elements may be fixed or may vary.

5.2.1. Syntax

An Array type specification is of the form:

Array [size] of ElementType

where ElementType is a declared type name or built-in type, and size is either the token * for a variable-sized array or an integer expression giving the number of elements for a fixed-size array.

5.2.2. Packed array type

The "packed" array type is like the "array" type, except that attempts are made to "pack" several small elements together where possible, instead of always aligning them on 16-bit boundaries. The syntax for the "packed array" type is the "Packed Array" keyword.

5.2.3. Examples

Some examples of Array specifications are:

```
Type
    Triple      = array [3] of Note;

Type
    Byte_Array  = packed array [*] of Byte;
```

5.2.4. Restrictions

Array elements must not be of pointer, variable-size array, or union types. Variable-size arrays can be declared; however, only pointers to the arrays can be directly passed in messages, along with a size parameter giving the number of elements being passed. The number of elements must be positive (greater than zero) for fixed-size arrays.

5.3. Enumeration Type

The Enumeration type provides a means of naming a set of related values and grouping those values together into a new type. The value of each name, if not specified, is provided by Matchmaker, beginning with zero and incrementing by one for each successive name.

5.3.1. Syntax

An Enumerated Type specification consists of a pair of parentheses enclosing a comma-separated list of enumeration elements of either the form:

ElementName

or:

ElementName = ElementValue

ElementNames without specified values are assigned sequential element values. Both kinds of elements may be mixed, as long as no value or name is duplicated.

5.3.2. Examples

Some examples of Enumeration specifications are:

```
Type
    Tri_State = (Open, Closed, Floating);

    Return_Values =
        (
        OK = 1,
        Information,
        Take_Heed,
        Warning = 99,
        Error = 1000,
        Fatal_Error = 10000,
        Internal_Error = -1
        );
```

5.3.3. Restrictions

All enumeration type element values must be expressible as 16-bit signed integers.

5.4. Pointer Types

Pointer types allow Matchmaker to pass a message by reference instead of directly including it into the message body. This is useful when passing variable-size arrays between processes, and occasionally for passing large fixed-size structures. (Do not use pointers to small structures, since at least one page of data is always passed regardless of the actual size.)

5.4.1. Syntax

A Pointer type specification is of the form:

↑ BaseType

where BaseType is a valid type specification.

5.4.2. Examples

Some examples of Pointer specifications are:

```
Type
    Bytes       = ↑ Byte_Array;
    Ports       = ↑ array [*] of Port;
    Block_Ptr   = ↑ Array [Block_Size] of Block_Elts;
```

5.4.3. Restrictions

Due to message passing restrictions, pointer base types must not be pointer types or union types.

5.5. Union Types

Union types provide a mechanism for passing data for which the ipc type number cannot be determined until run-time. This is useful only when the data is sometimes of a port type and sometimes a different port type, or is unstructured data. Only use a Union type when absolutely necessary.

5.5.1. Syntax

A Union type specification consists of a union head, of the form:

Union <TagType> of

followed by one or more fields of the form:

TagValue : (FieldName : FieldType)

terminated by the keywords End Union. The TagType must be an integer, character, boolean, or enumeration type. Each TagValue must be an expression of type TagType. Each FieldName must be a valid identifier, and each FieldType must be a declared type name or built-in type.

5.5.2. Examples

Some examples of Union specifications are:

```
Type
    PortRights =
      union <integer> of
        TypePtOwnership : (PtO: Port_Ownership);
        TypePtReceive   : (PtR: Port_Receive);
        TypePtAll       : (PtA: Port_All);
        TypePt          : (Pt:  Port);
      end union;

    Port_Or_Index =
      union <boolean> of
        False   : (Port_Index : long);
        True    : (Port_Value : port);
      end union;
```

5.5.3. Restrictions

Due to message passing restrictions, union field types must not be pointer types, variable-sized array types, or union types.

6. Messages/Remote Procedure Calls

Matchmaker provides several different kinds of calls that can be made between processes. These calls vary in the way errors are handled and in the directions in which the messages are sent.

A Matchmaker call is declared in a manner similar to Pascal or Ada procedures and functions: the introductory keyword specifies the class of call being declared, the call name is given, and then a parameter list. The call is usually terminated by a type or keyword giving the return value for the call.

6.1. Classes of Matchmaker Calls

For each of the individual Matchmaker declarations, CallName is an identifier giving the name of the message-based call being declared, and ArgList refers to a semicolon-separated list of call arguments.

6.1.1. Remote_Procedure calls

Remote_Procedure calls generate code that allows a user process to send request parameters to a server and to receive reply parameters back from the server.

Remote_Procedure declarations are of the form:

```
Remote_Procedure CallName ( ArgList ) : ValueType ;
```

If ValueType is the keyword GR_Value then CallName is a function with result type GeneralReturn (signed[16]). This is a success/error code. If any errors occur sending or receiving the messages for CallName, then CallName will return the send or receive error code.

If ValueType is the keyword No_Value then CallName is a procedure that does not return a value. If any errors occur sending or receiving the messages for CallName, then Matchmaker signals the send or receive error code in a language-dependent response.

If ValueType is neither GR_Value or No_Value then it must be a type name for a result type to be returned by CallName. In this case, CallName is a function returning a value of type ValueType. ValueType cannot be a variable-sized array or a union type. If any errors occur sending or receiving the messages for CallName, then Matchmaker signals the send or receive error code in a language-dependent response.

A RemotePort argument is required for all Remote_Procedure calls.

6.1.2. Message calls

Message calls generate code for a user process that sends a single message to a server without a reply.

Message declarations are of the form:

Message CallName (ArgList) : ValueType ;

If ValueType is the keyword GR_Value then CallName is a function with result type GeneralReturn (signed[16]) on the user side and a valueless procedure on the server side. When an error occurs sending the message for CallName, then CallName returns the send error code to the user. Otherwise it returns Success.

If ValueType is not GR_Value, then it must be the keyword No_Value. When ValueType is No_Value, then CallName is a procedure that does not return a value. If any errors occur sending the message for CallName, then Matchmaker signals the send error code in a language-dependent response.

A RemotePort argument is required for all Message calls. Only one-directional "In" parameters are allowed for Message calls (see 6.2, Call Arguments).

6.1.3. Server_Message calls

Server_Message calls generate code for a server process that sends a single message to a user process.

Server_Message declarations are of the form:

Server_Message CallName (ArgList) :ValueType ;

If ValueType is the keyword GR_Value then CallName is a function with result type GeneralReturn (signed[16]) on the server side and a valueless procedure on the user side. If any errors occur sending the message for CallName, CallName returns the send error code to the server. Otherwise it returns Success.

If ValueType is not GR_Value, then it must be the keyword No_Value. When ValueType is No_Value, then CallName is a procedure that does not return a value. If any errors occur sending the message for CallName, then Matchmaker signals the send error code in a language-dependent response.

A RemotePort argument is required for all Server_Message calls. Only one-directional "In" parameters are allowed for Server_Message calls.

6.1.4. Alternate_Reply calls

Alternate_Reply calls generate code for a server process that sends a reply message back to a user process in response to a Remote_Procedure that was different than expected. Alternate_Reply messages return values from exception conditions that occur during execution.

Alternate_Reply declarations forms are:

```
Alternate_Reply CallName ( ArgList ) ;

Alternate_Reply CallName ;
```

CallName is a language-dependent exception procedure that does not return a value.

A RemotePort argument is illegal for all Alternate_Reply calls. Only one-directional "In" parameters are allowed for Alternate_Reply calls.

6.2. Call Arguments

The Matchmaker call argument is:

```
ArgUsage ArgName : ArgType
```

where ArgUsage is a keyword specifying the argument to be used. ArgName is an identifier for the argument, and ArgType is the type of the argument.

ArgUsage can specify a direction for a data argument (for example, ArgUsage might be the keyword InOut), or it can be a keyword giving the special usage for the argument (such as the RemotePort keyword).

6.2.1. Data arguments

Most arguments to Matchmaker calls provide data that is passed between processes. Data arguments are of the form:

```
ArgDirection ArgParms : ArgType
```

6.2.1.1. Argument directions

ArgDirection specifies the direction(s) in which the data is sent. It may be one of In, Out, InOut, or may be omitted.

In arguments are only sent from the process initiating a call to the process handling the call.

Out — arguments are only sent from the process handing a call back to the process that initiated a call.

InOut — arguments are sent in both directions.

If ArgDirection is omitted, then In is assumed. Note: Some Matchmaker calls cannot accept Out or InOut arguments, because they specify single-directional exchanges.

6.2.1.2. Normal argument types and parameters

ArgType is the name of a built-in or declared Matchmaker type appropriate for message passing. New types may not be implicitly declared when declaring calls.

When ArgType does not specify a variable-size array or a union type, then ArgParms is an identifier corresponding to that argument. This is used for the corresponding target language parameter name for languages that require them.

An example of simple call arguments is given by the declaration:

```
Remote_Procedure(
  RemotePort    Server  : port;       ! (see 6.2.2)
  In            Put     : long;       ! In parameter
  Out           Get     : long;       ! Out parameter
  InOut         Both    : boolean;    ! InOut parameter
                Implied : character  ! In parameter
                )       : Gr_Value;
```

6.2.1.3. Variable-size array type parameters

When ArgType is the name for a pointer to a variable-size array (variable-size arrays cannot be passed; only pointers to them can be passed), then ArgParms must specify two parameter names. One is used to point to the array, and the other is used to dynamically indicate the number of elements in the array that is passed.

ArgParms for dynamic arrays can either be of the form:

PointerID [CountID]

or

[CountID] PointerID

The first is preferred. The only difference between the two forms is the order in which target language parameters for the pointer and count values are passed.

For example, the following Matchmaker declaration fragments:

```
type
    ByteVector = ↑ packed array [*] of byte;

Message Cattle(
                          : port;       ! (see 6.2.2)
in Bytes [Byte_Count] : ByteVector ! Dynamic array argument
     )                    : GR_Value;
```

would compile into the following C fragments:

```
typedef Byte ByteVector[];

GeneralReturn Cattle(ServPort, Bytes, Byte_Count);
    Port        ServPort;
    ByteVector  Bytes;              /* Pointer to array */
    long        Byte_Count;        /* Element count */
    {
    ...
    }
```

6.2.1.4. Union type parameters

When ArgType is the name for a union type, the ArgParms must specify two parameter names. One is the union tag (selector) value, and the other one passes the selected union field data.

Form of the ArgParms for union types:

DataID < TagID >

or

< TagID > DataID

The difference between the two forms is the order in which target language parameters for the field data and tag values are passed.

For example, the following Matchmaker declaration fragments:

```
type
    Port_Or_Index =
      union <boolean> of
        False    : (Port_Index : long);
        True     : (Port_Value : port);
      end union;

Remote_Procedure Wow(
                                : port;          ! (see 6.2.2)
out PortOrIndex <WhichOne>: Port_Or_Index ! Union argument
      )                         : No_Value;
```

would compile into the following Pascal fragments:

```
type
    Port_Or_Index =
      record case Boolean of
        False    : (Port_Index : Long);
        True     : (Port_Value : Port);
      end;

procedure Wow(
        ServPort        : port;
  var   PortOrIndex     : Port_Or_Index; { Data value }
  var   WhichOne        : Boolean);          { Tag value }
```

6.2.2. Special arguments

Several Matchmaker calls can dynamically provide information about how the messages implementing the call are sent and received. The syntax for each of them is of the form:

SpecialUsage ArgName : ArgType

The ArgName is an identifier for a parameter, and ArgType is the type name for that parameter. Specific ArgType values are required for each value of SpecialUsage.

SpecialUsage keywords:

RemotePort — Specify port to which request is sent. Required for all message classes except for Alternate_Reply. ArgType must be a port.

LocalPort — Specify port to which reply is sent. ArgType must be a port.

MsgType — Specify message type value for request message. Normal values are NormalMsg and EmergMsg (from Accent.mm). ArgType must be an integer. The default is Normal.

ReplyType — Specify message type value for reply message (for Remote_Procedure declarations). ReplyType is implicitly an Out parameter on the server side of an interface. Normal values are NormalMsg and EmergMsg (from Accent.mm). ArgType must be an integer. The default is Normal.

Send_Option Specify the send option field for the request message. Normal values are Wait, DontWait, and Reply (from Accent.mm). ArgType must be a 16-bit integer. The default is Wait.

Send_Timeout Specify the send timeout value for the request message. Values are in milliseconds, and a zero value means to wait indefinitely. ArgType must be a signed 32-bit integer. The default is Wait_Forever (0).

Receive_Timeout May be used to specify the receive timeout value for the reply message. Values are in milliseconds. A zero value means to wait indefinitely; -1 means return immediately if no message waiting. ArgType must be a signed 32-bit integer. The default is Wait_Forever (0).

A special abbreviation specifies the RemotePort argument for a Matchmaker call. If both SpecialUsage and ArgName are omitted, the call is interpreted the same as:

```
RemotePort ServPort : ArgType
```

6.3. Defaults and Options

Rather than passing special arguments to calls to specify every possible detail for every call, Matchmaker provides a set of defaults that cover most of the normal cases. These can generally be specified in one of two ways. Either the global default can be modified, or the default can be changed for a given message declaration.

6.3.1. Modifying global call defaults

The defaults for all messages in an interface can be changed using Options declarations. Options declarations must come before any data structure (Constant, Type, or Use) declarations.

An Options declaration consists of the Options keyword, followed by one or more declarations of the form:

```
OptionKey = OptionValue ;
```

(See the next chapter for more on Options declarations.)

6.3.2. Modifying local call defaults

The defaults for any particular call can be changed by inserting one or more clauses of the form:

```
, OptionKey = OptionValue
```

before the semicolon (;) terminating the call declaration. Such a clause is only effective for that specific call.

6.3.3. Listing of defaults

The set of call defaults which can be changed are a subset of the call special arguments. A list of the keys, defaults, and legal values are:

Key	Default	Legal Values
MsgType	NormalMsg (= 0)	NormalMsg, EmergencyMsg (0, 1)
ReplyType	NormalMsg (= 0)	NormalMsg, EmergencyMsg (0, 1)
Send_Option	Wait (= 0)	Wait, DontWait, Reply (0..2)
Send_Timeout	0 (Infinity) (in milliseconds)	32-bit signed integers
Receive_Timeout	0 (Infinity) (in milliseconds)	32-bit signed integers

7. Options and Message ID Declarations

7.1. Options Declarations

The Options declaration is used to change the default values of Matchmaker parameters. Options declarations must come before any data structure (Constant, Type, or Use) declarations.

Options declarations only affect the specification in which they occur. Another specification that references the specification containing the Options declarations (for example, by a Use declaration) is not affected by the declarations.

7.1.1. Syntax

An Options declaration consists of the Options keyword and one or more declarations of the form:

OptionKey = OptionValue ;

7.1.2. Non-call argument options

Options that use special arguments were listed in the previous chapter. Other options are explained here:

Protocol_Version — Declare version number of code generator algorithm to be used. Currently only version 1 is supported.

Local_Ports — Declare maximum number of ports that are allocated on the user side for outstanding calls to the server. A value of 0 requires all calls to specify a LocalPort special parameter. A value of * specifies no limit. Default is 1.

Ports_Backlog — Backlog value for any Local_Ports allocated. Default is 0 (requesting Accent default Backlog value).

7.1.3. Examples

```
Options
    Protocol_Version = 1;
    Local_Ports = 5;
    Receive_Timeout = 100;      ! Be impatient with it.
```

7.2. Message ID Declarations

Calls are normally assigned message ID numbers sequentially, in ascending order from the declared Interface ID number. Reply IDs for Remote_Procedure, Alternate_Reply, and Server_Message calls have 100 added to them, to distinguish them from request IDs.

There are instances, however, where it might be necessary to assign the IDs in another fashion. Matchmaker

provides two declarations strictly for manipulating the message ID numbers that are assigned to declared calls.

7.2.1. Skip_ID declaration

The Skip_ID declaration increments the message ID counter exactly as a call declaration would. The primary purpose of the Skip_ID declaration is to replace an obsolete call declaration in an interface file.

The syntax for Skip_ID is:

```
Skip_ID ;
```

7.2.2. Next_ID declaration

The Next_ID declaration is used to directly set the message ID counter to a new value. The new value should be within 100 of the initial value.

The syntax for Next_ID is:

```
Next_ID = NewValue ;
```

where NewValue is an integer expression.

8. Using Matchmaker

This chapter describes how to use Matchmaker and code generated by Matchmaker.

8.1. Using a Generated Interface

Matchmaker only generates code that allows parameters for kinds of calls to be passed between User and Server processes. The code implementing the calls and any code that uses the calls must be hand-coded.

This section describes the specifics of writing code that is called from Matchmaker procedures and code that calls Matchmaker procedures. This section assumes that an interface called Prog has been declared.

For a detailed example, please refer Section 11.

8.1.1. Using Matchmaker user code

The following applies to code written for the user side of a Matchmaker interface:

- The procedure InitProg must be called before any other procedure from module ProgUser (user side code) to initialize ProgUser.
- Any Message or Remote_Procedure can be called directly using appropriate target language calling conventions.
- Code must be written by the user for any Alternate_Reply calls declared to handle the Alternate_Reply and take appropriate actions when they occur.
- The ProgDispatcher function may be called to decode parameters and to call the function implementing any Server_Message procedures declared in Prog. ProgDispatcher is passed a message. It returns true for messages it recognizes, and false for those it does not.
- Code must be written by the user for each declared Server_Message call that takes appropriate action for the Server_Message. These routines are called by the ProgDispatcher function for appropriate messages.

8.1.2. Using Matchmaker server code

The following applies to code written for the server side of a Matchmaker interface:

- The implementor must write code to receive the message and then call the ProgServer function to decode message parameters and to call the function implementing any Message or Remote_Procedure procedures declared in Prog for received messages. ProgServer must be passed both the message received and a reply message. It returns a true for messages it recognizes and false for those it does not. The reply message is sent unless the RetValue field of the call is NOREPLY.
- The implementor must write code for each declared Message and the declared Remote_Procedure call that implements the call. These routines are called by the ProgServer function for appropriate messages.
- An Alternate_Reply message can reply to a Remote_Procedure request by signaling the

Alternate_Reply flag with appropriate parameters. Construct an alternate reply message for ProgServer to return.

- Any Server_Message may be directly called via appropriate target language calling conventions to pack and send the message.

8.2. Matchmaker File Names

The source file name for a Matchmaker specification named Prog should be:

Prog.mm

If a set of language-specific declarations for the specification are needed for a given language that normally uses the extension .ext, then those declarations should be in the file:

Prog.mm-ext

When compiled for a language using extension .ext, Matchmaker will then normally produce the files:

- ProgDefs.ext (Data structures) ProgUser.ext (User side code) ProgServer.ext (Server side code)

For example, the Time interface uses the following files:

Time.mm	Matchmaker source
TimeDefs.pas	Pascal types for time TimeUser.pas Pascal user interface to time server TimeServer.pas Pascal server interface for time server (The Time server is implemented in Pascal)
Time.h	C types for time TimeUser.c C user interface to time server
TimeDefs.slisp	Accent Lisp types for time TimeMsgDefs Accent Lisp types for time messages TimeUser.slisp Accent Lisp user interface to time server TimeUInit Load-time initialization forms for the time interface

If language-specific declarations are needed, they are specified in the files:

- Time.mm-pas (Pascal-specific declarations) Time.mm-c (C-specific declarations) Time.mm-slisp (Accent Lisp-specific declarations)

Information specific to each language is discussed in Section 10.

8.3. Matchmaker Command Line Syntax

Invoke Matchmaker with the command line:

Matchmaker FileName -Lang1 = Opt1 ... -LangN = OptN

where FileName is the name of the specification file to be compiled without the .mm extension. The Lang names are currently members of the set Pascal, C, and Accent Lisp. The Opt values must be members of the set:

All	Generate all files for Lang Defs
	Generate Defs file for Lang User
	Generate User file for Lang Server
	Generate Server file for Lang NoUser
	Generate all but User file for Lang NoServer
	Generate all but Server file for Lang

The order of switches and the filename does not matter. Unique abbreviations for both switches and options are accepted.

Other switches are:

Help	Display information about Matchmaker Verbose
	Display progress information Quiet
	Do not display progress information Errorfile
	Generate a file of error messages NoErrorfile
	Do not generate a file of error messages Loop
	Allow additional specification files to be processed before exiting

9. A BNF for the Matchmaker Language

The following is a BNF description of the Matchmaker language. Conventions used are as follows:

- Literal tokens are enclosed in double quotes ("").
- Optional productions are enclosed in square brackets ([]).
- Three periods (...) denote optional repetition.
- A vertical bar (|) denotes choices between productions.
- Braces ({}) enclose a group of required productions.
- Parens(()) enclose comments.

9.1. Interface and Options Definitions

```
Interface               ::= Interface_Decl
                                [Options_Decl]...
                                [Data_Decl]...
                                [Msg_Decl]...
                                End_Interface |
                            Types_Decl
                                [Options_Decl]...
                                [Data_Decl]...
                                End_Types

Interface_Decl          ::= "Interface" Interface_Name "="
                            Msg_ID_Base ";"

Types_Decl              ::= "Types" Interface_Name ";"

Interface_Name          ::= Identifier

Msg_ID_Base             ::= Integer_Constant

Reply_Base              ::= Integer_Constant

End_Interface           ::= "End" "Interface"

End_Types               ::= "End" "Types"

Options_Decl            ::= "Options" {Option_Decl ";"}...
                            (not yet implemented)

Option_Decl ::= Msg_Options | Msg_Name_Options |
                Protocol_Options

Msg_Name_Options  ::= Msg_Name_Key "=" Identifier
                            (not yet implemented)

Msg_Name_Ke     ::= "Server_Prefix" | "User_Prefix"

Protocol_Options  ::= "Protocol_Version" = Integer_Constant
                            (not yet implemented)
```

9.2. Datatype Definitions

```
Data_Decl     ::= Use_Decl | Type_Decl | Constant_Decl

Use_Decl       ::= "Use" Single_Use...

Single_Use     ::= Interface_Name "From" File_Name ";"

File_Name       ::= String_Constant

Constant_Decl   ::= "Constant" Single_Constant...

Single_Constant ::= Constant_Name "=" Constant_Expr ";"

Constant_Name   ::= Identifier

Type_Decl       ::= "Type" Single_Type...

Single_Type    ::= Type Name "=" Type_Specification
                     ["," Type_Option]... ";"

Type_Name          ::= Identifier

Type_Specification  ::= Type_Name | Builtin_Type |
          Array_Type | Record_Type |Pointer_Type |
                  Enumeration_Type | Union_Type

Builtin_Type      ::= "Boolean" | "Character" | "Real" |
               Integer_Type | String_Type | Port_Type

Integer_Type   ::= "Unsigned" ["[" Integer_Constant "]"] |
                    "Signed" ["[" Integer_Constant "]"] |
                         Subrange_Type | "Long" | "Short" |
                         "Byte"

Subrange_Type    ::= Integer_Constant ".." Integer_Constant

Port_Type     ::= "Port" | "Port_Send" | "Port_Receive" |
                    "Port_Ownership" | "Port_All"

String_Type     ::= "PERQ_String" ["[" Integer_Constant "]"]


Array_Type      ::= [Packing] "Array" "[" Array_Size "]"
                        "Of" Type_Specification

Array_Size      ::= Integer_Constant | "*"

Packing          ::= "Packed" | "Unpacked"
```

```
Record_Type     ::= [Packing] "Record" Record_Component...
                        "End" "Record"

Record_Component
              ::= Field_Identifier ":" Type_Specification ";"

Field_Identifier     ::= Identifier

Pointer_Type         ::= "↑" Type_Specification

Enumeration_Type     ::= "(" Enum_List ")"

Enum_List  ::= Implicit_Enum_List | Explicit_Enum_List

Implicit_Enum_List ::= Enum_Name ["," Enum_Name]...

Explicit_Enum_List ::= Enum_Element ["," Enum_Element]...

Enum_Element ::= Enum_Name ["=" Integer_Constant]

Enum_Name       ::= Identifier

Union_Type   ::= "Union: "<" Union_Selector_Type ">"
                   "Of" Union_Component... "End" "Union"

Union_Selector_Type  ::= Type_Specification

Union_Component ::= Union_Tag ":" "(" [Record_Component]
                              ")" ";"

Union_Tag      ::= Constant_Expr | "Otherwise"

Type_Option   ::= "TypeType" "=" Integer_Constant |
                      "Deallocate" ["=" Boolean_Constant] |
                      "NoDeallocate" |
                      "Element_Size" "=" Integer_Constant |
                      "Element_Count" "=" Integer_Constant
```

9.3. Message Definitions

```
Msg_Decl ::= Msg_Code_Decl | Msg_ID_Decl

Msg_Code_Decl ::= Msg_Body ["," Msg_Options]... ";"

Msg_Body   ::= To_Server | From_Server |
                 Remote_Procedure | Alternate_Reply

Msg_Options ::= Msg_Param_Key "=" Integer_Constant |
            Msg_CodeGen_Key "=" Boolean_Constant (NYI)

To_Server ::= "Message" Arg_List ":" Msg_Result
```

```
Remote_Procedure::= "Remote_Procedure" Arg_List ":" Msg_Result

Alternate_Reply  ::= "Alternate_Reply" [Arg_List]

From_Server      ::= "Server_Message" Arg_List

Arg_List         ::= "(" Msg_Arg [";" Msg_Arg]... ")"

Msg_Arg          ::= Data_Arg | Special_Arg

Data_Arg          ::= [Arg_Direction] Data_Arg_Spec;

Arg_Direction    ::= "In" | "Out" | "InOut"

Data_Arg_Spec ::= Simple_Arg_Spec | Variable_Arg_Spec |
                      Union_Arg_Spec

Simple_Arg_Spec      ::= Arg_Name ":" Arg_Type

Variable_Arg_Spec    ::= Arg_Name "[" Arg_Cnt_Name "]" ":" Arg_Type |
                         "[" Arg_Cnt_Name "]" Arg_Name ":" Arg_Type

Union_Arg_Spec       ::= Arg_Name "<" Selector_Name ">" ":" Arg_Type |
                         "<" Selector_Name ">" Arg_Name ":" Arg_Type

Special_Arg  ::= Special_Usage Arg_Name ":" Arg_Type |
                      ":" Arg_Type

Special_Usage ::= Port_Usage_Key | Msg_Param_Key

Port_Usage_Key       ::= "RemotePort" | "LocalPort"

Msg_Param_Key        ::= "MsgType" | "Send_Option" |
                         "Send_Timeout" | "Receive_Timeout"

Msg_CodeGen_Key      ::= "Asynchronous" ["User" | "Server"]

Arg_Cnt_Name         ::= Arg_Name

Selector_Name        ::= Arg_Name

Arg_Name             ::= Identifier

Arg_Type             ::= Type_Name [ "," Type_Option]...

Msg_ID_Decl          ::= "Skip_ID" ";" |
                         "Next_ID" "=" Integer_Constant ";"
```

9.4. Expression Syntax

```
Constant_Expr         ::= OR_CTCE (Valid types context dependent)

Integer_Constant      ::= Adding_CTCE (Must be integer valued)

Boolean_Constant      ::= Or_CTCE (Must be boolean valued)

Character_Constant    ::= Primary_CTCE (Must be character valued)

String_Constant       ::= Primary_CTCE (Must be string valued)

Enumeration_Constant ::= Primary_CTCE (Must result in a declared
                             Enum_Name identifier)

Or_CTCE               ::= And_CTCE ["Or" And_CTCE]...

And_CTCE              ::= Not_CTCE ["And" Not_CTCE]...

Not_CTCE              ::= ["Not"] Relational_CTCE

Relational_CTCE       ::= Equality_CTCE
                             [ {">" | ">=" | "<=" | "<" }
                             Equality_CTCE ]...

Equality_CTCE ::= Adding_CTCE [ {"=" | "<>"} Adding_CTCE]...

Adding_CTCE           ::= [{"+" | "-"}] Multiplying_CTCE
                             [{{"+" | "-"} Multiplying_CTCE]...

Multiplying_CTCE      ::= Primary CTCE
                             [{"*" | "/" | "Mod"} Primary CTCE]...

Primary_CTCE          ::= Identifier | Constant_Lexeme |
                             "(" Or_CTCE ")"
```

9.5. Lexical Definitions

```
Constant_Lexeme  ::= Octal_Literal | Decimal_Literal |
                     String_Literal | Character_Literal |
                     Boolean_Literal

Octal_Literal    ::= "#" followed by a non-empty octal
                     digit string

Decimal_Literal   ::= A non-empty decimal digit string

String_Literal    ::= A character string enclosed in double
                      quotes. A double quote may be
                      represented by two.

Character_Literal    ::= A character enclosed in single quotes.
                         A single quote may be represented
                         by two.

Boolean_Literal      ::= "True" | "False"

Identifier       ::= A string composed of letters, digits
                     the underscore character, not starting
                     with a digit. Identifiers are matched
                     in a non-case-sensitive manner.
```

At any lexical break, comments can be inserted as:

"!" comment text <End_Of_Line>

10. Information Specific to Each Code Generator

10.1. Pascal

To import files into a Matchmaker-generated Pascal user or server file, create a file named

FOO.mm-pas

where FOO is the name of the interface declared at the beginning of the Matchmaker declaration file. The contents of FOO.mm-pas should look like:

```
(in-package "MM-PASCAL-GEN")
(simports "file1")
(simports "file2")
(uimports "file3")
(uimports "file4")
(imports "file5")
(imports "file6")
```

In this example the files file1.pas, file2.pas, file5.pas, and file6.pas will be imported into FOOServer.pas, and files number 3, 4, 5, and 6 will be imported into FOOUser.pas.

10.2. C

Using Matchmaker with the C language works the same as with Pascal except:

1. (In-package "MM-PASCAL-GEN") becomes (in-package "MM-C-GEN").
2. Filenames have the extension .c instead of .pas.
3. If the module name and filename are not the same, you should state both, as in:

   (imports "ModuleName" "FileName")

   Otherwise, Matchmaker assumes the module and file have the same name.

10.3. Lisp

Using Matchmaker with the Accent Lisp language works the same as with Pascal except:

1. The source file in the above example would be named FOO.mm-slisp.
2. The contents of the source file would be:

   (in-package "MM-LISP-GEN")
   <-arbitrary Lisp expressions to be evaluated->)

For more information, see the *Accent Lisp Manual.*

11. Example

In the example given in this section, Matchmaker generates the Typescript client interface for Pascal and Lisp and the Typescript server interface for Pascal.

11.1. Source Files

11.1.1. Typescript.mm

This file is a language-independent interface specification.

```
interface TS=2800;
!*************************    ****************************************
!
! Author:       Pedro Szekely
! Abstract:     Matchmaker interface for Typescript
!
!--------------------
! Change log:
!  Aug 27 1984  jrg     Oops!  Had to change TSCharArray arg
!                       STSPutCharArray to
!                       a pTSCharArray, of course.
!  Jul 31 1984  jrg     Changed ptschararray arg to STSPutCharArray to
!                       a TSCharArray with a char_count arg for
!                       new MM format.
!                       Also changed def of TSCharArray to be
!                       variable-sized for backwards compatibility.
!  May 25 1984  JmL     Converted to new matchmaker format.
!  Nov  3 1983  dbg     Change STSFlushOutput to a Procedure.  Now it
!                       passes back a message to the user, so it can
!                       be used to synchronize Typescript with other
!                       operations on the typescript window.
!  Oct 28 1983  pas     Rearranged STSPutCharArray to avoid
!                       recompilations
!  Oct 24 1983  pas     Added STSPutCharArray, TSCharArray
!  Oct 17 1983  dbg     Use allocated server message range (2800).
!  Sep 6  1983  dbg     Import TSDefs on user side; TSTypes on server
!  Sep 3  1983  pas     Add OpenWindow, FullOpenWindow, SetEnv,
!                       GrabWindow
!  Aug 23 1983  pas     Add GetString, PutString
!  Aug 9  1983  pas     Created
!-------------------------------------------------------------------


use Accint from "accent";
use Sapphdefs from "sapphdefs";
```

```
type
        Typescript    = Port;
        TSString255   = Perq_String[255];
        TSCharArray   = packed array[*] of Character;
        pTSCharArray  = ↑ TSCharArray;

Remote_Procedure STSOpen(: Port; vp: ViewPort; env: Port): Typescript;

Remote_Procedure STSOpenWindow( : Port;
                        w: Window;
                        env: Port
                        ): Typescript;

Remote_Procedure STSFullOpen(    :      Port;
                        vp :            ViewPort;
                        env :           Port;
                        fontName :      TSString255;
                        doWrap :        Boolean;
                        dispPages :     Integer
                        ): Typescript;

Remote_Procedure STSFullOpenWindow( :      Port;
                        w :             Window;
                        env :           Port;
                        fontName :      TSString255;
                        doWrap :        Boolean;
                        dispPages :     Integer
                        ): Typescript;

Remote_Procedure STSFullLine(: Typescript): Boolean;

Remote_Procedure STSGetChar(: Typescript): Character;

Remote_Procedure STSGetString(: Typescript): TSString255;

Message STSPutChar(: Typescript; ch: Character):No_Value;

Message STSPutString(: Typescript; s: TSString255):No_Value;

Message STSFlushInput(: Typescript):No_Value;
```

```
Remote_Procedure STSFlushOutput(: Typescript):No_Value;
                                        !use for synchronization

Message STSChangeEnv(: Typescript; env: Port):No_Value;

Remote_Procedure STSGrabWindow(: Typescript; kPort: Port): Window;

Message STSPutCharArray(: Typescript;
                                chars [char_count]: pTSCharArray;
                                firstCh: Integer;
                                lastCh: Integer
                                ):No_Value;

end interface
```

11.1.2. Ts.mm-pas

This file contains Pascal-specific declarations.

```
(in-package "MM-PASCAL-GEN")
(simports "TSTypes")
(simports "STypescript")
```

11.2. Matchmaker-Produced Files

11.2.1. TsDefs.pas

This file contains Pascal types.

```
Module TSDefs;

Exports

Imports BuiltinDefs from BuiltinDefs;
Imports AccintDefs from AccintDefs;
Imports SdefsDefs from SdefsDefs;

type
   Typescript = port;
   TSString255 = string[255];
   TSCharArray = packed array [0 .. 0] of char;
   pTSCharArray = ↑TSCharArray;

Private

Procedure Bug;
begin end.
```

11.2.2. TsUser.pas

This file is the Pascal user interface.

```
Module TS;

{ -------> } Exports { <------- }

Imports AccInt from AccintUser;
Imports TSDefs from TSDefs;

Procedure InitTS(Rport : port);

Function STSOpen(
    ServPort : Port;
    vp : Viewport;
    env : port): Typescript;
Function STSOpenWindow(
    ServPort : Port;
    w : Window;
    env : port): Typescript;

Function STSFullOpen(
    ServPort : Port;
    vp : Viewport;
    env : port;
    fontName : TSString255;
    doWrap : boolean;
    dispPages : Integer): Typescript;

Function STSFullOpenWindow(
    ServPort : Port;
    w : Window;
    env : port;
    fontName : TSString255;
    doWrap : boolean;
    dispPages : Integer): Typescript;

Function STSFullLine(
    ServPort : Port): boolean;

Function STSGetChar(
    ServPort : Port): char;

Function STSGetString(
    ServPort : Port): TSString255;

Procedure STSPutChar(
    ServPort : Port;
    ch : char);
```

```
Procedure STSPutString(
    ServPort : Port;
    s : TSString255);

Procedure STSFlushInput(
    ServPort : Port);

Procedure STSFlushOutput(
    ServPort : Port);

Procedure STSChangeEnv(
    ServPort : Port;
    env : port);

Function STSGrabWindow(
    ServPort : Port;
    kPort : port): Window;

Procedure STSPutCharArray(
    ServPort : Port;
    chars : pTSCharArray;
    char_count : long;
    firstCh : Integer;
    lastCh : Integer);

Function TSAsynch(InP: Pointer): boolean;

{ -------> } Private { <------- }

Imports AccCall from AccCall;
Imports PascalInit from PascalInit;
type
 DumMsg = record
  head : Msg;
  body: array [0..1023] of integer
 end;
 ptrDumMsg = ↑DumMsg;

var
 ReplyPort : Port;
 GR        : GeneralReturn;
 RoP : ptrDumMsg;

Procedure TSExceptions;
 forward;
```

```
Procedure InitTS(Rport : port);
begin
 if RPort = NullPort then
  begin
   GR:=AllocatePort(KernelPort,ReplyPort,5);
   if GR<>Success then ReplyPort:=DataPort
  end
  else ReplyPort:=RPort;
 new(ReP)
end;


Function STSOpen(
    ServPort : Port;
    vp : Viewport;
    env : port): Typescript;

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : Viewport;
  IPCNam3 : TypeType;
  Arg3    : port
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam4 : TypeType;
  Arg4    : Typescript
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;
```

```
begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 38;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2800
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 6;
   IPCNam2.TypeSizeInBits := 32;
   IPCNam2.NumObjects := 1;
   Arg2 := vp;
   IPCNam3.InLine := TRUE;
   IPCNam3.Deallocate := FALSE;
   IPCNam3.LongForm := FALSE;
   IPCNam3.TypeName := 6;
   IPCNam3.TypeSizeInBits := 32;
   IPCNam3.NumObjects := 1;
   Arg3 := env;
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSOpen)
  end;
 GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
 If GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSOpen)
  end;
```

```
 with RepMsgP↑ do
  begin
   If head.ID <> 2900 then
    begin
     TSExceptions;
     exit(STSOpen)
    end;
   if RetCodeType.TypeName <> TYPEINT16 then
    begin
     Raise GRError(BADREPLY);
     exit(STSOpen)
    end;
   if (RetCode <> Success) then
    begin
     Raise GRError(BadReply);
     Exit(STSOpen)
    end;
   if IPCNam4.TypeName <> 6 then
    begin
     Raise GRError(BadReply);
     exit(STSOpen)
    end;
   STSOpen := Arg4;
  end;
end;

Function STSOpenWindow(
    ServPort : Port;
    w : Window;
    env : port): Typescript;

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : Window;
  IPCNam3 : TypeType;
  Arg3    : port
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam4 : TypeType;
  Arg4    : Typescript
 end;
 ptrRepMsg  = ↑RepMessage;
```

```
var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 38;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2801
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 6;
   IPCNam2.TypeSizeInBits := 32;
   IPCNam2.NumObjects := 1;
   Arg2 := w;
   IPCNam3.InLine := TRUE;
   IPCNam3.Deallocate := FALSE;
   IPCNam3.LongForm := FALSE;
   IPCNam3.TypeName := 6;
   IPCNam3.TypeSizeInBits := 32;
   IPCNam3.NumObjects := 1;
   Arg3 := env;
  end;
```

```
with RepMsgP↑.head do
 begin
  MsgSize := WordSize(DumMsg)*2;
  LocalPort := ReplyPort
 end;
GR := Send(MyMsg.head,0,WAIT);
if GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSOpenWindow)
 end;
GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSOpenWindow)
 end;
with RepMsgP↑ do
 begin
  If head.ID <> 2901 then
   begin
    TSExceptions;
    exit(STSOpenWindow)
   end;
  if RetCodeType.TypeName <> TYPEINT16 then
   begin
    Raise GRError(BADREPLY);
    exit(STSOpenWindow)
   end;
  if (RetCode <> Success) then
   begin
    Raise GRError(BadReply);
    Exit(STSOpenWindow)
   end;
  if IPCNam4.TypeName <> 6 then
   begin
    Raise GRError(BadReply);
    exit(STSOpenWindow)
   end;
  STSOpenWindow := Arg4;
 end;
end;
```

```
Function STSFullOpen(
    ServPort : Port;
    vp : Viewport;
    env : port;
    fontName : TSString255;
    doWrap : boolean;
    dispPages : Integer): Typescript;

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : Viewport;
  IPCNam3 : TypeType;
  Arg3    : port;
  IPCNam4 : TypeType;
  Arg4    : TSString255;
  IPCNam5 : TypeType;
  Arg5    : boolean;
  IPCNam6 : TypeType;
  Arg6    : Integer
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam7 : TypeType;
  Arg7    : Typescript
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;
```

```
begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 310;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2802
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 6;
   IPCNam2.TypeSizeInBits := 32;
   IPCNam2.NumObjects := 1;
   Arg2 := vp;
   IPCNam3.InLine := TRUE;
   IPCNam3.Deallocate := FALSE;
   IPCNam3.LongForm := FALSE;
   IPCNam3.TypeName := 6;
   IPCNam3.TypeSizeInBits := 32;
   IPCNam3.NumObjects := 1;
   Arg3 := env;
   IPCNam4.InLine := TRUE;
   IPCNam4.Deallocate := FALSE;
   IPCNam4.LongForm := FALSE;
   IPCNam4.TypeName := 0;
   IPCNam4.TypeSizeInBits := 8;
   IPCNam4.NumObjects := 256;
   Arg4 := fontName;
   IPCNam5.InLine := TRUE;
   IPCNam5.Deallocate := FALSE;
   IPCNam5.LongForm := FALSE;
   IPCNam5.TypeName := 0;
   IPCNam5.TypeSizeInBits := 1;
   IPCNam5.NumObjects := 1;
   Arg5 := doWrap;
   IPCNam6.InLine := TRUE;
   IPCNam6.Deallocate := FALSE;
   IPCNam6.LongForm := FALSE;
   IPCNam6.TypeName := 1;
   IPCNam6.TypeSizeInBits := 16;
   IPCNam6.NumObjects := 1;
   Arg6 := dispPages;
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
```

```
GR := Send(MyMsg.head,0,WAIT);
if GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSFullOpen)
 end;
GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSFullOpen)
 end;
with RepMsgP↑ do
 begin
  If head.ID <> 2902 then
   begin
    TSExceptions;
    exit(STSFullOpen)
   end;
  if RetCodeType.TypeName <> TYPEINT16 then
   begin
    Raise GRError(BADREPLY);
    exit(STSFullOpen)
   end;
  if (RetCode <> Success) then
   begin
    Raise GRError(BadReply);
    Exit(STSFullOpen)
   end;
  if IPCNam7.TypeName <> 6 then
   begin
    Raise GRError(BadReply);
    exit(STSFullOpen)
   end;
  STSFullOpen := Arg7;
 end;
end;
```

```
Function STSFullOpenWindow(
    ServPort : Port;
    w : Window;
    env : port;
    fontName : TSString255;
    doWrap : boolean;
    dispPages : Integer): Typescript;

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : Window;
  IPCNam3 : TypeType;
  Arg3    : port;
  IPCNam4 : TypeType;
  Arg4    : TSString255;
  IPCNam5 : TypeType;
  Arg5    : boolean;
  IPCNam6 : TypeType;
  Arg6    : Integer
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam7 : TypeType;
  Arg7    : Typescript
 end;
 ptrRepMsg  = ↑RepMessage;
```

```
var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 310;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2803
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 6;
   IPCNam2.TypeSizeInBits := 32;
   IPCNam2.NumObjects := 1;
   Arg2 := w;
   IPCNam3.InLine := TRUE;
   IPCNam3.Deallocate := FALSE;
   IPCNam3.LongForm := FALSE;
   IPCNam3.TypeName := 6;
   IPCNam3.TypeSizeInBits := 32;
   IPCNam3.NumObjects := 1;
   Arg3 := env;
   IPCNam4.InLine := TRUE;
   IPCNam4.Deallocate := FALSE;
   IPCNam4.LongForm := FALSE;
   IPCNam4.TypeName := 0;
   IPCNam4.TypeSizeInBits := 8;
   IPCNam4.NumObjects := 256;
   Arg4 := fontName;
   IPCNam5.InLine := TRUE;
   IPCNam5.Deallocate := FALSE;
   IPCNam5.LongForm := FALSE;
   IPCNam5.TypeName := 0;
   IPCNam5.TypeSizeInBits := 1;
   IPCNam5.NumObjects := 1;
   Arg5 := doWrap;
   IPCNam6.InLine := TRUE;
   IPCNam6.Deallocate := FALSE;
   IPCNam6.LongForm := FALSE;
   IPCNam6.TypeName := 1;
   IPCNam6.TypeSizeInBits := 16;
   IPCNam6.NumObjects := 1;
   Arg6 := dispPages;
  end;
```

```
with RepMsgP↑.head do
 begin
  MsgSize := WordSize(DumMsg)*2;
  LocalPort := ReplyPort
 end;
GR := Send(MyMsg.head,0,WAIT);
if GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSFullOpenWindow)
 end;
GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSFullOpenWindow)
 end;
with RepMsgP↑ do
 begin
  If head.ID <> 2903 then
   begin
    TSExceptions;
    exit(STSFullOpenWindow)
   end;
  if RetCodeType.TypeName <> TYPEINT16 then
   begin
    Raise GRError(BADREPLY);
    exit(STSFullOpenWindow)
   end;
  if (RetCode <> Success) then
   begin
    Raise GRError(BadReply);
    Exit(STSFullOpenWindow)
   end;
  if IPCNam7.TypeName <> 6 then
   begin
    Raise GRError(BadReply);
    exit(STSFullOpenWindow)
   end;
  STSFullOpenWindow := Arg7;
 end;
end;
```

```
Function STSFullLine(
    ServPort : Port): boolean;

type
 MyMessage = record
  head    : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam2 : TypeType;
  Arg2    : boolean
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;
begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 22;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2804
  end;
 with MyMsg do
  begin
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSFullLine)
  end;
 GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
```

```
If GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSFullLine)
 end;
with RepMsgP↑ do
 begin
  If head.ID <> 2904 then
   begin
    TSExceptions;
    exit(STSFullLine)
   end;
  if RetCodeType.TypeName <> TYPEINT16 then
   begin
    Raise GRError(BADREPLY);
    exit(STSFullLine)
   end;
  if (RetCode <> Success) then
   begin
    Raise GRError(BadReply);
    Exit(STSFullLine)
   end;
  if IPCNam2.TypeName <> 0 then
   begin
    Raise GRError(BadReply);
    exit(STSFullLine)
   end;
  STSFullLine := Arg2;
 end;
end;

Function STSGetChar(
    ServPort : Port): char;

type
 MyMessage = record
  head    : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam2 : TypeType;
  Arg2    : char
 end;
 ptrRepMsg  = ↑RepMessage;
```

```
var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 22;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2806
  end;
 with MyMsg do
  begin
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSGetChar)
  end;
 GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
 If GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSGetChar)
  end;
```

```
 with RepMsgP↑ do
  begin
   If head.ID <> 2905 then
    begin
     TSExceptions;
     exit(STSGetChar)
    end;
   if RetCodeType.TypeName <> TYPEINT16 then
    begin
     Raise GRError(BADREPLY);
     exit(STSGetChar)
    end;
   if (RetCode <> Success) then
    begin
     Raise GRError(BadReply);
     Exit(STSGetChar)
    end;
   if IPCNam2.TypeName <> 8 then
    begin
     Raise GRError(BadReply);
     exit(STSGetChar)
    end;
   STSGetChar := Arg2;
  end;
end;

Function STSGetString(
    ServPort : Port): TSString255;

type
 MyMessage = record
  head    : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam2 : TypeType;
  Arg2    : TSString255
 end;
 ptrRepMsg  = ↑RepMessage;
```

```
var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 22;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2806
  end;
 with MyMsg do
  begin
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSGetString)
  end;
 GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
 If GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSGetString)
  end;
```

```
 with RepMsgP↑ do
  begin
   If head.ID <> 2906 then
    begin
     TSExceptions;
     exit(STSGetString)
    end;
   if RetCodeType.TypeName <> TYPEINT16 then
    begin
     Raise GRError(BADREPLY);
     exit(STSGetString)
    end;
   if (RetCode <> Success) then
    begin
     Raise GRError(BadReply);
     Exit(STSGetString)
    end;
   if IPCNam2.TypeName <> 0 then
    begin
     Raise GRError(BadReply);
     exit(STSGetString)
    end;
   STSGetString := Arg2;
  end;
end;

Procedure STSPutChar(
    ServPort : Port;
    ch : char);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : char
 end;
type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;
```

```
begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 28;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2807
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 8;
   IPCNam2.TypeSizeInBits := 8;
   IPCNam2.NumObjects := 1;
   Arg2 := ch;
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSPutChar)
  end;
end;
```

```
Procedure STSPutString(
    ServPort : Port;
    s : TSString256);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : TSString256
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 282;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2808
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 0;
   IPCNam2.TypeSizeInBits := 8;
   IPCNam2.NumObjects := 256;
   Arg2 := s;
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSPutString)
  end;
end;
```

```
Procedure STSFlushInput(
    ServPort : Port);

type
 MyMessage = record
  head    : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 22;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2809
  end;
 with MyMsg do
  begin
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSFlushInput)
  end;
end;
```

```
Procedure STSFlushOutput(
    ServPort : Port);

type
 MyMessage = record
  head    : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
 ptrRepMsg  = ↑RepMessage;
```

```
var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := TRUE;
   MsgSize:= 22;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2810
  end;
 with MyMsg do
  begin
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSFlushOutput)
  end;
 GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
 If GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSFlushOutput)
  end;
 with RepMsgP↑ do
  begin
   If head.ID <> 2910 then
    begin
     TSExceptions;
     exit(STSFlushOutput)
    end;
   if RetCodeType.TypeName <> TYPEINT16 then
    begin
     Raise GRError(BADREPLY);
     exit(STSFlushOutput)
    end;
   if (RetCode <> Success) then
    begin
     Raise GRError(BadReply);
     Exit(STSFlushOutput)
    end;
  end;
end;
```

```
Procedure STSChangeEnv(
    ServPort : Port;
    env : port);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : port
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 30;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2811
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 6;
   IPCNam2.TypeSizeInBits := 32;
   IPCNam2.NumObjects := 1;
   Arg2 := env;
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSChangeEnv)
  end;
end;
```

```
Function STSGrabWindow(
    ServPort : Port;
    kPort : port): Window;

type
 MyMessage = record
  head     : Msg;
  IPCNam2  : TypeType;
  Arg2     : port
 end;

type
 RepMessage = record
  head         : Msg;
  RetCodeType  : TypeType;
  RetCode      : GeneralReturn;
  IPCNam3  : TypeType;
  Arg3     : Window
 end;
 ptrRepMsg   = ↑RepMessage;
```

```
var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;

begin
 RepMsgP := RECAST(ReP,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 30;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2812
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 6;
   IPCNam2.TypeSizeInBits := 32;
   IPCNam2.NumObjects := 1;
   Arg2 := kPort;
  end;
 with RepMsgP↑.head do
  begin
   MsgSize := WordSize(DumMsg)*2;
   LocalPort := ReplyPort
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSGrabWindow)
  end;
```

```
GR := Receive(RepMsgP↑.head,0,LOCALPT,RECEIVEIT);
If GR <> Success then
 begin
  Raise GRError(GR);
  exit(STSGrabWindow)
 end;
with RepMsgP↑ do
 begin
  If head.ID <> 2912 then
   begin
    TSExceptions;
    exit(STSGrabWindow)
   end;
  if RetCodeType.TypeName <> TYPEINT16 then
   begin
    Raise GRError(BADREPLY);
    exit(STSGrabWindow)
   end;
  if (RetCode <> Success) then
   begin
    Raise GRError(BadReply);
    Exit(STSGrabWindow)
   end;
  if IPCNam3.TypeName <> 6 then
   begin
    Raise GRError(BadReply);
    exit(STSGrabWindow)
   end;
  STSGrabWindow := Arg3;
 end;
end;
```

```
Procedure STSPutCharArray(
    ServPort : Port;
    chars : pTSCharArray;
    char_count : long;
    firstCh : Integer;
    lastCh : Integer);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  TName2  : integer;
  TSize2  : integer;
  NumElts2: long;
  Arg2    : pTSCharArray;
  IPCNam3 : TypeType;
  Arg3    : Integer;
  IPCNam4 : TypeType;
  Arg4    : Integer
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
 ptrRepMsg  = ↑RepMessage;

var
 MyMsg : MyMessage;
 RepMsgP : ptrRepMsg;
```

```
begin
 RepMsgP := RECAST(RePort,ptrRepMsg);
 with MyMsg.head do
  begin
   Simplemsg := FALSE;
   MsgSize:= 50;
   MsgType:= NormalMsg;
   RemotePort := ServPort;
   LocalPort := ReplyPort;
   ID := 2813
  end;
 with MyMsg do
  begin
   IPCNam2.InLine := FALSE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := TRUE;
   TName2 := 8;
   TSize2 := 8;
   NumElts2 := char_count;
   Arg2 := chars;
   IPCNam3.InLine := TRUE;
   IPCNam3.Deallocate := FALSE;
   IPCNam3.LongForm := FALSE;
   IPCNam3.TypeName := 1;
   IPCNam3.TypeSizeInBits := 16;
   IPCNam3.NumObjects := 1;
   Arg3 := firstCh;
   IPCNam4.InLine := TRUE;
   IPCNam4.Deallocate := FALSE;
   IPCNam4.LongForm := FALSE;
   IPCNam4.TypeName := 1;
   IPCNam4.TypeSizeInBits := 16;
   IPCNam4.NumObjects := 1;
   Arg4 := lastCh;
  end;
 GR := Send(MyMsg.head,0,WAIT);
 if GR <> Success then
  begin
   Raise GRError(GR);
   exit(STSPutCharArray)
  end;
end;
```

```
Function TSAsynch(InP : POINTER): boolean;
var
 InMsgP  : ptrMsg;

begin
 InMsgP := RECAST(InP,ptrMsg);
 with InMsgP↑ do
  begin
   TSAsynch := True;
   case shrink(ID) of
    Otherwise: TSAsynch := False
   end
  end
end;


Procedure TSExceptions;
begin
 with RePˆ.head do
  case shrink(ID) of
   Otherwise: raise GRError(BADREPLY)
  end
end.
```

11.2.3. TsServer.pas

This file is the Pascal server interface.

```
Module TSServer;

{ -------> } Exports { <------- }

Imports AccInt from AccintUser;
Imports TSDefs from TSDefs;
Imports STypescript from STypescript;
Imports TSTypes from TSTypes;

Function TSServer(InP,RepP : POINTER) : boolean;

{ -------> } Private { <------- }

Imports AccCall from AccCall;
Imports PascalInit from PascalInit;

var
 ReplyPort: port;
 GR       : GeneralReturn;
 ReP      : Pointer;


Procedure XSTSOpen(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg;
  IPCNam2 : TypeType;
  Arg2    : Viewport;
  IPCNam3 : TypeType;
  Arg3    : port
 end;
type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam4 : TypeType;
  Arg4    : Typescript
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 vp : Viewport;
 env : port;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := FALSE;
   MsgSize := 36;
   MsgType := NormalMsg;
   ID := 2900
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSOpen)
    end;
   vp := Arg2;
   if IPCNam3.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSOpen)
    end;
   env := Arg3;
  end;
 RepMsgP↑.Arg4 := STSOpen(
    ServPort,
    vp,
    env);
 with RepMsgP↑ do
  begin
   IPCNam4.InLine := TRUE;
   IPCNam4.Deallocate := FALSE;
   IPCNam4.LongForm := FALSE;
   IPCNam4.TypeName := 6;
   IPCNam4.TypeSizeInBits := 32;
   IPCNam4.NumObjects := 1;
  end
end;

Procedure XSTSOpenWindow(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg;
  IPCNam2 : TypeType;
  Arg2    : Window;
  IPCNam3 : TypeType;
  Arg3    : port
 end;
```

```
type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam4 : TypeType;
  Arg4    : Typescript
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 w : Window;
 env : port;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := FALSE;
   MsgSize := 36;
   MsgType := NormalMsg;
   ID := 2901
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSOpenWindow)
    end;
   w := Arg2;
   if IPCNam3.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSOpenWindow)
    end;
   env := Arg3;
  end;
 RepMsgP↑.Arg4 := STSOpenWindow(
    ServPort,
    w,
    env);
 with RepMsgP↑ do
  begin
   IPCNam4.InLine := TRUE;
   IPCNam4.Deallocate := FALSE;
   IPCNam4.LongForm := FALSE;
   IPCNam4.TypeName := 6;
   IPCNam4.TypeSizeInBits := 32;
   IPCNam4.NumObjects := 1;
  end
end;
```

```
Procedure XSTSFullOpen(InP, ReP : POINTER);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : Viewport;
  IPCNam3 : TypeType;
  Arg3    : port;
  IPCNam4 : TypeType;
  Arg4    : TSString256;
  IPCNam5 : TypeType;
  Arg5    : boolean;
  IPCNam6 : TypeType;
  Arg6    : Integer
 end;
type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam7 : TypeType;
  Arg7    : Typescript
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 vp : Viewport;
 env : port;
 fontName : TSString256;
 doWrap : boolean;
 dispPages : Integer;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(RePtrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := FALSE;
   MsgSize := 36;
   MsgType := NormalMsg;
   ID := 2902
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpen)
    end;
   vp := Arg2;
   if IPCNam3.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpen)
    end;
   env := Arg3;
   if IPCNam4.TypeName <> 0 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpen)
    end;
   fontName := Arg4;
   if IPCNam5.TypeName <> 0 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpen)
    end;
   doWrap := Arg5;
   if IPCNam6.TypeName <> 1 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpen)
    end;
   dispPages := Arg6;
  end;
 RepMsgP↑.Arg7 := STSFullOpen(
    ServPort,
    vp,
    env,
    fontName,
    doWrap,
    dispPages);
```

```
 with RepMsgP↑ do
  begin
   IPCNam7.InLine := TRUE;
   IPCNam7.Deallocate := FALSE;
   IPCNam7.LongForm := FALSE;
   IPCNam7.TypeName := 6;
   IPCNam7.TypeSizeInBits := 32;
   IPCNam7.NumObjects := 1;
  end
end;

Procedure XSTSFullOpenWindow(InP, ReP : POINTER);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : Window;
  IPCNam3 : TypeType;
  Arg3    : port;
  IPCNam4 : TypeType;
  Arg4    : TSString255;
  IPCNam5 : TypeType;
  Arg5    : boolean;
  IPCNam6 : TypeType;
  Arg6    : Integer
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam7 : TypeType;
  Arg7    : Typescript
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 w : Window;
 env : port;
 fontName : TSString255;
 doWrap : boolean;
 dispPages : Integer;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := FALSE;
   MsgSize := 36;
   MsgType := NormalMsg;
   ID := 2903
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpenWindow)
    end;
   w := Arg2;
   if IPCNam3.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpenWindow)
    end;
   env := Arg3;
   if IPCNam4.TypeName <> 0 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpenWindow)
    end;
   fontName := Arg4;
   if IPCNam5.TypeName <> 0 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpenWindow)
    end;
   doWrap := Arg5;
   if IPCNam6.TypeName <> 1 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSFullOpenWindow)
    end;
   dispPages := Arg6;
  end;
```

```
 RepMsgP↑.Arg7 := STSFullOpenWindow(
    ServPort,
    w,
    env,
    fontName,
    doWrap,
    dispPages);
 with RepMsgP↑ do
  begin
   IPCNam7.InLine := TRUE;
   IPCNam7.Deallocate := FALSE;
   IPCNam7.LongForm := FALSE;
   IPCNam7.TypeName := 6;
   IPCNam7.TypeSizeInBits := 32;
   IPCNam7.NumObjects := 1;
  end
end;

Procedure XSTSFullLine(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam2 : TypeType;
  Arg2    : boolean
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := TRUE;
   MsgSize := 34;
   MsgType := NormalMsg;
   ID := 2904
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
  end;
 RepMsgP↑.Arg2 := STSFullLine(
    ServPort);
 with RepMsgP↑ do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 0;
   IPCNam2.TypeSizeInBits := 1;
   IPCNam2.NumObjects := 1;
  end
end;

Procedure XSTSGetChar(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam2 : TypeType;
  Arg2    : char
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(RoP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := TRUE;
   MsgSize := 34;
   MsgType := NormalMsg;
   ID := 2905
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
  end;
 RepMsgP↑.Arg2 := STSGetChar(
    ServPort);
 with RepMsgP↑ do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 8;
   IPCNam2.TypeSizeInBits := 8;
   IPCNam2.NumObjects := 1;
  end
end;

Procedure XSTSGetString(InP, RoP : POINTER);

type
 MyMessage = record
  head   : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam2 : TypeType;
  Arg2    : TSString255
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := TRUE;
   MsgSize := 288;
   MsgType := NormalMsg;
   ID := 2906
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
  end;
 RepMsgP↑.Arg2 := STSGetString(
    ServPort);
 with RepMsgP↑ do
  begin
   IPCNam2.InLine := TRUE;
   IPCNam2.Deallocate := FALSE;
   IPCNam2.LongForm := FALSE;
   IPCNam2.TypeName := 0;
   IPCNam2.TypeSizeInBits := 8;
   IPCNam2.NumObjects := 256;
  end
end;

Procedure XSTSPutChar(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg;
  IPCNam2 : TypeType;
  Arg2    : char
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 ch : char;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 8 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSPutChar)
    end;
   ch := Arg2;
  end;
 STSPutChar(
    ServPort,
    ch);
 RepMsgP↑.RetCode := NoReply
end;

Procedure XSTSPutString(InP, ReP : POINTER);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : TSString255
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 s : TSString255;
```

```
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 0 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSPutString)
    end;
   s := Arg2;
  end;
 STSPutString(
    ServPort,
    s);
 RepMsgP↑.RetCode := NoReply
end;

Procedure XSTSFlushInput(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;

begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
  end;
 STSFlushInput(
    ServPort);
 RepMsgP↑.RetCode := NoReply
end;
```

```
Procedure XSTSFlushOutput(InP, ReP : POINTER);

type
 MyMessage = record
  head    : Msg
 end;

type
 RepMessage = record
  head         : Msg;
  RetCodeType : TypeType;
  RetCode      : GeneralReturn
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;

begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := TRUE;
   MsgSize := 28;
   MsgType := NormalMsg;
   ID := 2910
  end;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
  end;
 STSFlushOutput(
    ServPort);
 with RepMsgP↑ do
  begin
  end
end;
```

```
Procedure XSTSChangeEnv(InP, ReP : POINTER);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  Arg2    : port
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 env : port;
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSChangeEnv)
    end;
   env := Arg2;
  end;
 STSChangeEnv(
    ServPort,
    env);
 RepMsgP↑.RetCode := NoReply
end;
```

```
Procedure XSTSGrabWindow(InP, ReP : POINTER);

type
 MyMessage = record
  head   : Msg;
  IPCNam2 : TypeType;
  Arg2    : port
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
  IPCNam3 : TypeType;
  Arg3    : Window
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 kPort : port;

begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with RepMsgP↑.head do
  begin
   Simplemsg := FALSE;
   MsgSize := 36;
   MsgType := NormalMsg;
   ID := 2912
  end;
```

```
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if IPCNam2.TypeName <> 6 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSGrabWindow)
    end;
   kPort := Arg2;
  end;
 RepMsgP↑.Arg3 := STSGrabWindow(
    ServPort,
    kPort);
 with RepMsgP↑ do
  begin
   IPCNam3.InLine := TRUE;
   IPCNam3.Deallocate := FALSE;
   IPCNam3.LongForm := FALSE;
   IPCNam3.TypeName := 6;
   IPCNam3.TypeSizeInBits := 32;
   IPCNam3.NumObjects := 1;
  end
end;

Procedure XSTSPutCharArray(InP, ReP : POINTER);

type
 MyMessage = record
  head    : Msg;
  IPCNam2 : TypeType;
  TName2  : integer;
  TSize2  : integer;
  NumElts2: long;
  Arg2    : pTSCharArray;
  IPCNam3 : TypeType;
  Arg3    : Integer;
  IPCNam4 : TypeType;
  Arg4    : Integer
 end;

type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn
 end;
type
 ptrMyMsg  = ↑MyMessage;
 ptrRepMsg = ↑RepMessage;
```

```
var
 MyMsgP  : ptrMyMsg;
 RepMsgP : ptrRepMsg;
 ServPort : Port;
 chars : pTSCharArray;
 char_count : long;
 firstCh : Integer;
 lastCh : Integer;
begin
 MyMsgP := RECAST(InP,ptrMyMsg);
 RepMsgP := RECAST(ReP,ptrRepMsg);
 RepMsgP↑.RetCode := Success;
 with MyMsgP↑ do
  begin
   ServPort := head.LocalPort;
   if TName2 <> 8 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSPutCharArray)
    end;
   char_count := NumElts2;
   chars := Arg2;
   if IPCNam3.TypeName <> 1 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSPutCharArray)
    end;
   firstCh := Arg3;
   if IPCNam4.TypeName <> 1 then
    begin
     RepMsgP↑.RetCode := WRONGARGS;
     exit(XSTSPutCharArray)
    end;
   lastCh := Arg4;
  end;
 STSPutCharArray(
    ServPort,
    chars,
    char_count,
    firstCh,
    lastCh);
 RepMsgP↑.RetCode := NoReply
end;
```

```
Function TSServer(InP, RepP : POINTER): boolean;
type
 RepMessage = record
  head        : Msg;
  RetCodeType : TypeType;
  RetCode     : GeneralReturn;
 end;
 ptrRepMessage = ↑RepMessage;
var
 InMsgP  : ptrMsg;
 RepMsgP : ptrRepMessage;

begin
 InMsgP  := RECAST(InP,ptrMsg);
 RepMsgP := RECAST(RepP,ptrRepMessage);
 with RepMsgP↑.Head do
  begin
   LocalPort  := InMsgP↑.LocalPort;
   RemotePort := InMsgP↑.RemotePort
  end;
 with RepMsgP↑.RetCodeType do
  begin
   TypeName := TYPEINT16;
   TypeSizeInBits := 16;
   NumObjects := 1;
   InLine := true;
   LongForm := false;
   Deallocate := false
  end;
 TSServer := True;
 with InMsgP↑ do
  case shrink(ID) of
   2800: XSTSOpen(InP,RepP);
   2801: XSTSOpenWindow(InP,RepP);
   2802: XSTSFullOpen(InP,RepP);
   2803: XSTSFullOpenWindow(InP,RepP);
   2804: XSTSFullLine(InP,RepP);
   2805: XSTSGetChar(InP,RepP);
   2806: XSTSGetString(InP,RepP);
   2807: XSTSPutChar(InP,RepP);
   2808: XSTSPutString(InP,RepP);
   2809: XSTSFlushInput(InP,RepP);
   2810: XSTSFlushOutput(InP,RepP);
   2811: XSTSChangeEnv(InP,RepP);
   2812: XSTSGrabWindow(InP,RepP);
   2813: XSTSPutCharArray(InP,RepP);
   Otherwise: begin
               TSServer := False;
               RepMsgP↑.RetCode := BADMSGID
              end
  end
end.
```

11.2.4. TsDefs.slisp

This file contains Accent Lisp types.

```
;;;;   -*-Lisp-*-
;;;;
;;;;  Matchmaker generated types and constants definitions file for TS
;;;;
;;;;
       ;;;;            THIS FILE SHOULD NOT BE HAND EDITED
;;;;
;;;;  To change this interface, edit the interface matchmaker
;;;;  file and run it through matchmaker to generate this file.
;;;;


(in-package "TSDEFS")
(use-package "MMINTERNALDEFS")
(use-package "BUILTINDEFS")

(use-package "ACCINTDEFS")

(use-package "SAPPHDEFSDEFS")


(defoperator (Typescript-op port) ((alien port)) ' (alien-value , alien))

(defmacro access-Typescript (alien)
  '
  (alien-access (Typescript-op , alien) port))


(defoperator (TSString255-op simple-string) ((alien (perq-string 255))) '
             (alien-value , alien))

(defmacro access-TSString255 (alien)
  '
  (alien-access (TSString255-op , alien) simple-string))

(defmacro access-TSCharArray (a i)
  '
  (alien-access (index-TSCharArray , a , i) string-char))


(defoperator (index-TSCharArray string-char) ((a TSCharArray) i) '
             (alien-index (alien-value , a) (* , i 16) 8))

(defoperator (deref-pTSCharArray TSCharArray)
             ((p (ref TSCharArray)) size-in-bits) '
             (alien-indirect (alien-value , p) , size-in-bits))

(defmacro address-pTSCharArray (alien)
  '
  (alien-access , alien system-area-pointer))

(export
 '(pTSCharArray address-pTSCharArray deref-pTSCharArray TSCharArray
  index-TSCharArray access-TSCharArray TSString255 access-TSString255
  TSString255-op Typescript access-Typescript Typescript-op))
```

11.2.5. TsUser.slisp

This file is the Accent Lisp user interface.

```
;;;;   -*-Lisp-*-
;;;;
;;;;  Matchmaker generated user interface file for TS
;;;;
;;;;
;;;;            THIS FILE SHOULD NOT BE HAND EDITED
;;;;
;;;;  To change this interface, edit the interface matchmaker
;;;;  file and run it through matchmaker to generate this file.
;;;;



(in-package "TSUSER")
(use-package "TSDEFS")
(use-package "BUILTINDEFS")
(use-package "MMINTERNALDEFS")


(use-package "ACCINTUSER")
(defvar *receiveport* NIL)
(defconstant interface-backlog 0)

(use-package "SAPPHDEFSDEFS")

(use-package "ACCINTDEFS")


;;; TS-Send -- internal
;;;
;;; Send macro for TS
;;;
(defmacro TS-Send (msg argblock wait-time send-option)
  "User send macro for interface TS"
  '
  (cond ((fixnump *receiveport*)
         (alien-store (msg-localport-op , msg) port *receiveport*)
         (send , argblock , wait-time , send-option))
        (t
         notaport)))

;;;
;;; User side interface initialization function.
;;;
(defun TS-Init (user-port)
  "The user side intialization function"
  (cond ((eql user-port nullport)
    (multiple-value-bind (gr port) (allocateport kernelport interface-backlog)
          (if (eql gr success) (setq *receiveport* port)
              (setq *receiveport* dataport))
          gr))
        (t
         (setq *receiveport* user-port)
         success)))
```

```
;;; STSOpen -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSOpen (Remote_Port vp env)
  "The user side of remote procedure STSOpen"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSOpen to-STSOpen))
            (alien-store
            (msg-remoteport-op (Msgize-to-STSOpen-op (alien-value to))) port
                    Remote_Port)
            (alien-store (to-STSOpen-vp-op (alien-value to)) port vp)
            (alien-store (to-STSOpen-env-op (alien-value to)) port env)
            (setq gr
                  (TS-send (Msgize-to-STSOpen-op (alien-value to)) to-STSOpen
                   send-waittime send-option)))
 (alien-bind ((from from-STSOpen from-STSOpen))
            (cond ((eql gr success)
                   (alien-store
                    (msg-msgsize-op
                     (Msgize-from-STSOpen-op (alien-value from)))
                    (signed-byte 32) 36)
                   (alien-store
                    (msg-localport-op
                     (Msgize-from-STSOpen-op (alien-value from)))
                    port *ReceivePort*)
                   (setq gr
                         (receive from-STSOpen receive-waittime localpt
                                  receivelt))
                   (cond ((not (eql gr success))
                          NIL
                          (return gr))
                         ((=
                           (access-msg-id
                            (Msgize-from-STSOpen-op (alien-value from)))
                           2900)
                          (cond ((=
                                  (access-typetypei-typename
                                   (msg-retcode-typetypei-op
                                       (Msgize-from-STSOpen-op
                                     (alien-value from))))
                                  1)
                                 (setq gr
                                       (access-msg-retcode
                                        (Msgize-from-STSOpen-op
                                         (alien-value from)))))
                                (t
                                 NIL
                                 (return badreply)))
```

```
                        (cond ((eql
                                (access-typetypei-typename
                                 (from-STSOpen-R_e_s_u_l_t-typetypeI-op
                                  (alien-value from)))
                                6)
                               (setq R_e_s_u_l_t
                                     (access-from-STSOpen-R_e_s_u_l_t
                                      (alien-value from))))
                              (t
                               NIL
                               (return BADREPLY)))
                        NIL
                        (return (values R_e_s_u_l_t)))
                       (t
                        NIL
                        (return
                         (user-alternate-return
                          (Msgize-from-STSOpen-op (alien-value from)))))))
                 (t
                  NIL
                  (return gr))))))
;;; STSOpenWindow -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSOpenWindow (Remote_Port w env)
  "The user side of remote procedure STSOpenWindow"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSOpenWindow to-STSOpenWindow))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSOpenWindow-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSOpenWindow-w-op (alien-value to)) port w)
                (alien-store (to-STSOpenWindow-env-op (alien-value to)) port env)
                (setq gr
                      (TS-send (Msgize-to-STSOpenWindow-op (alien-value to))
                       to-STSOpenWindow send-waittime send-option)))
    (alien-bind ((from from-STSOpenWindow from-STSOpenWindow))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSOpenWindow-op (alien-value from)))
                        (signed-byte 32) 36)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSOpenWindow-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSOpenWindow receive-waittime localpt
                                      receiveit))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSOpenWindow-op (alien-value from)))
```

```
          2901)
         (cond ((=
                 (access-typetypei-typename
                  (msg-retcode-typetypei-op
                   (Msgize-from-STSOpenWindow-op
                    (alien-value from))))
                 1)
                (setq gr
                      (access-msg-retcode
                       (Msgize-from-STSOpenWindow-op
                        (alien-value from)))))
               (t
                NIL
                (return badreply)))
         (cond ((eql
                 (access-typetypei-typename
                  (from-STSOpenWindow-R_e_s_u_l_t-typetypeI-op
                   (alien-value from)))
                 6)
                (setq R_e_s_u_l_t
                      (access-from-STSOpenWindow-R_e_s_u_l_t
                       (alien-value from))))
               (t
                NIL
                (return BADREPLY)))
         NIL
         (return (values R_e_s_u_l_t)))
        (t
         NIL
         (return
          (user-alternate-return
           (Msgize-from-STSOpenWindow-op (alien-value from)))))))
(t
 NIL
 (return gr))))))
```

```
;;; STSFullOpen -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSFullOpen (Remote_Port vp env fontName doWrap dispPages)
  "The user side of remote procedure STSFullOpen"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSFullOpen to-STSFullOpen))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSFullOpen-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSFullOpen-vp-op (alien-value to)) port vp)
                (alien-store (to-STSFullOpen-env-op (alien-value to)) port env)
                (alien-store (to-STSFullOpen-fontName-op (alien-value to))
                             simple-string fontName)
                (alien-store (to-STSFullOpen-doWrap-op (alien-value to)) boolean
                             doWrap)
                (alien-store (to-STSFullOpen-dispPages-op (alien-value to))
                             (signed-byte 16) dispPages)
                (setq gr
                      (TS-send (Msgize-to-STSFullOpen-op (alien-value to))
                       to-STSFullOpen send-waittime send-option)))
    (alien-bind ((from from-STSFullOpen from-STSFullOpen))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSFullOpen-op (alien-value from)))
                        (signed-byte 32) 36)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSFullOpen-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSFullOpen receive-waittime localpt
                                      receiveit))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSFullOpen-op (alien-value from)))
                               2902)
```

```
            (cond ((=
                    (access-typetypei-typename
                     (msg-retcode-typetypei-op
                      (Msgize-from-STSFullOpen-op
                       (alien-value from))))
                    1)
                   (setq gr
                         (access-msg-retcode
                          (Msgize-from-STSFullOpen-op
                           (alien-value from)))))
                  (t
                   NIL
                   (return badreply)))
            (cond ((eql
                    (access-typetypei-typename
                     (from-STSFullOpen-R_e_s_u_l_t-typetypeI-op
                      (alien-value from)))
                    6)
                   (setq R_e_s_u_l_t
                         (access-from-STSFullOpen-R_e_s_u_l_t
                          (alien-value from))))
                  (t
                   NIL
                   (return BADREPLY)))
            NIL
            (return (values R_e_s_u_l_t)))
           (t
            NIL
            (return
             (user-alternate-return
              (Msgize-from-STSFullOpen-op (alien-value from)))))))
    (t
     NIL
     (return gr))))))
```

```
;;; STSFullOpenWindow -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSFullOpenWindow (Remote_Port w env fontName doWrap dispPages)
  "The user side of remote procedure STSFullOpenWindow"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSFullOpenWindow to-STSFullOpenWindow))
                (alien-store
                 (msg-remoteport-op
                  (Msgize-to-STSFullOpenWindow-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSFullOpenWindow-w-op (alien-value to)) port w)
                (alien-store (to-STSFullOpenWindow-env-op (alien-value to)) port
                             env)
                (alien-store (to-STSFullOpenWindow-fontName-op (alien-value to))
                             simple-string fontName)
                (alien-store (to-STSFullOpenWindow-doWrap-op (alien-value to))
                             boolean doWrap)
                (alien-store (to-STSFullOpenWindow-dispPages-op (alien-value to))
                             (signed-byte 16) dispPages)
                (setq gr
                      (TS-send (Msgize-to-STSFullOpenWindow-op (alien-value to))
                       to-STSFullOpenWindow send-waittime send-option)))
    (alien-bind ((from from-STSFullOpenWindow from-STSFullOpenWindow))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSFullOpenWindow-op (alien-value from)))
                        (signed-byte 32) 36)
```

```
(alien-store
 (msg-localport-op
  (Msgize-from-STSFullOpenWindow-op (alien-value from)))
 port *ReceivePort*)
(setq gr
      (receive from-STSFullOpenWindow receive-waittime
               localpt receivelt))
(cond ((not (eql gr success))
       NIL
       (return gr))
      ((=
        (access-msg-id
         (Msgize-from-STSFullOpenWindow-op
          (alien-value from)))
        2903)
       (cond ((=
               (access-typetypei-typename
                (msg-retcode-typetypei-op
                 (Msgize-from-STSFullOpenWindow-op
                  (alien-value from))))
               1)
              (setq gr
                    (access-msg-retcode
                     (Msgize-from-STSFullOpenWindow-op
                      (alien-value from)))))
             (t
              NIL
              (return badreply)))
       (cond ((eql
               (access-typetypei-typename
                (from-STSFullOpenWindow-R_e_s_u_l_t-typetypeI-op
                 (alien-value from)))
               6)
              (setq R_e_s_u_l_t
                    (access-from-STSFullOpenWindow-R_e_s_u_l_t
                     (alien-value from))))
             (t
              NIL
              (return BADREPLY)))
       NIL
       (return (values R_e_s_u_l_t)))
      (t
       NIL
       (return
        (user-alternate-return
         (Msgize-from-STSFullOpenWindow-op
          (alien-value from)))))))
(t
 NIL
 (return gr))))))
```

```
;;; STSFullLine -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSFullLine (Remote_Port)
  "The user side of remote procedure STSFullLine"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSFullLine to-STSFullLine))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSFullLine-op (alien-value to)))
                 port Remote_Port)
                (setq gr
                      (TS-send (Msgize-to-STSFullLine-op (alien-value to))
                       to-STSFullLine send-waittime send-option)))
    (alien-bind ((from from-STSFullLine from-STSFullLine))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSFullLine-op (alien-value from)))
                        (signed-byte 32) 34)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSFullLine-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSFullLine receive-waittime localpt
                                      receiveit))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSFullLine-op (alien-value from)))
                               2904)
```

```
            (cond ((=
                    (access-typetypei-typename
                     (msg-retcode-typetypei-op
                      (Msgize-from-STSFullLine-op
                       (alien-value from))))
                    1)
                   (setq gr
                         (access-msg-retcode
                          (Msgize-from-STSFullLine-op
                           (alien-value from)))))
                  (t
                   NIL
                   (return badreply)))
            (cond ((eql
                    (access-typetypei-typename
                     (from-STSFullLine-R_e_s_u_l_t-typetypeI-op
                      (alien-value from)))
                    0)
                   (setq R_e_s_u_l_t
                         (access-from-STSFullLine-R_e_s_u_l_t
                          (alien-value from))))
                  (t
                   NIL
                   (return BADREPLY)))
            NIL
            (return (values R_e_s_u_l_t)))
           (t
            NIL
            (return
             (user-alternate-return
              (Msgize-from-STSFullLine-op (alien-value from)))))))
    (t
     NIL
     (return gr))))))
```

```
;;; STSGetChar -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSGetChar (Remote_Port)
  "The user side of remote procedure STSGetChar"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSGetChar to-STSGetChar))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSGetChar-op (alien-value to)))
                 port Remote_Port)
                (setq gr
                      (TS-send (Msgize-to-STSGetChar-op (alien-value to))
                       to-STSGetChar send-waittime send-option)))
    (alien-bind ((from from-STSGetChar from-STSGetChar))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSGetChar-op (alien-value from)))
                        (signed-byte 32) 34)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSGetChar-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSGetChar receive-waittime localpt
                                      receiveit))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSGetChar-op (alien-value from)))
                               2905)
                              (cond ((=
                                      (access-typetypei-typename
                                       (msg-retcode-typetypei-op
                                        (Msgize-from-STSGetChar-op
                                         (alien-value from))))
                                      1)
                                     (setq gr
                                           (access-msg-retcode
                                            (Msgize-from-STSGetChar-op
                                             (alien-value from)))))
                                    (t
                                     NIL
                                     (return badreply)))
                              (cond ((eql
                                      (access-typetypei-typename
                                       (from-STSGetChar-R_e_s_u_l_t-typetypeI-op
                                        (alien-value from)))
                                      8)
                                     (setq R_e_s_u_l_t
                                           (access-from-STSGetChar-R_e_s_u_l_t
                                            (alien-value from))))
                                    (t
                                     NIL
                                     (return BADREPLY)))
                              NIL
                              (return (values R_e_s_u_l_t)))
```

```
                        (t
                         NIL
                         (return
                          (user-alternate-return
                           (Msgize-from-STSGetChar-op (alien-value from)))))))
                    (t
                     NIL
                     (return gr))))))


;;; STSGetString -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSGetString (Remote_Port)
  "The user side of remote procedure STSGetString"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSGetString to-STSGetString))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSGetString-op (alien-value to)))
                 port Remote_Port)
                (setq gr
                      (TS-send (Msgize-to-STSGetString-op (alien-value to))
                       to-STSGetString send-waittime send-option)))
    (alien-bind ((from from-STSGetString from-STSGetString))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSGetString-op (alien-value from)))
                        (signed-byte 32) 288)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSGetString-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSGetString receive-waittime localpt
                                      receivelt))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSGetString-op (alien-value from)))
                               2906)
```

```
          (cond ((=
                  (access-typetypei-typename
                   (msg-retcode-typetypei-op
                    (Msgize-from-STSGetString-op
                     (alien-value from))))
                  1)
                 (setq gr
                       (access-msg-retcode
                        (Msgize-from-STSGetString-op
                         (alien-value from)))))
                (t
                 NIL
                 (return badreply)))
          (cond ((eql
                  (access-typetypei-typename
                   (from-STSGetString-R_e_s_u_l_t-typetypeI-op
                    (alien-value from)))
                  0)
                 (setq R_e_s_u_l_t
                       (access-from-STSGetString-R_e_s_u_l_t
                        (alien-value from))))
                (t
                 NIL
                 (return BADREPLY)))
          NIL
          (return (values R_e_s_u_l_t)))
         (t
          NIL
          (return
           (user-alternate-return
            (Msgize-from-STSGetString-op (alien-value from)))))))
(t
 NIL
 (return gr))))))
```

```
;;; STSPutChar -- public
;;;
;;; User-side message interface.
;;;
(defun STSPutChar (Remote_Port ch)
  "The user side of message STSPutChar"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSPutChar to-STSPutChar))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSPutChar-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSPutChar-ch-op (alien-value to)) string-char
                             ch))
    (send to-STSPutChar send-maxwait send-option)))


;;; STSPutString -- public
;;;
;;; User-side message interface.
;;;
(defun STSPutString (Remote_Port s)
  "The user side of message STSPutString"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSPutString to-STSPutString))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSPutString-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSPutString-s-op (alien-value to))
                             simple-string s))
    (send to-STSPutString send-maxwait send-option)))


;;; STSFlushInput -- public
;;;
;;; User-side messags interface.
;;;
(defun STSFlushInput (Remote_Port)
  "The user side of message STSFlushInput"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSFlushInput to-STSFlushInput))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSFlushInput-op (alien-value to)))
                 port Remote_Port))
    (send to-STSFlushInput send-maxwait send-option)))
```

```
;;; STSFlushOutput -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSFlushOutput (Remote_Port)
  "The user side of remote procedure STSFlushOutput"
  (prog ((send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSFlushOutput to-STSFlushOutput))
                (alien-store
                 (msg-remoteport-op
                  (Msgize-to-STSFlushOutput-op (alien-value to)))
                 port Remote_Port)
                (setq gr
                      (TS-send (Msgize-to-STSFlushOutput-op (alien-value to))
                       to-STSFlushOutput send-waittime send-option)))
    (alien-bind ((from from-STSFlushOutput from-STSFlushOutput))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSFlushOutput-op (alien-value from)))
                        (signed-byte 32) 28)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSFlushOutput-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSFlushOutput receive-waittime
                                      localpt receiveit))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSFlushOutput-op (alien-value from)))
                               2910)
                              (cond ((=
                                      (access-typetypei-typename
                                       (msg-retcode-typetypei-op
                                        (Msgize-from-STSFlushOutput-op
                                         (alien-value from))))
                                      1)
                                     (setq gr
                                           (access-msg-retcode
                                            (Msgize-from-STSFlushOutput-op
                                             (alien-value from)))))
                                    (t
                                     NIL
                                     (return badreply)))
                              NIL
                              (return (values)))
                             (t
                              NIL
                              (return
                               (user-alternate-return
                                (Msgize-from-STSFlushOutput-op (alien-value from)))))))
                      (t
                       NIL
                       (return gr))))))
```

```
;;; STSChangeEnv -- public
;;;
;;; User-side message interface.
;;;
(defun STSChangeEnv (Remote_Port env)
  "The user side of message STSChangeEnv"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSChangeEnv to-STSChangeEnv))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSChangeEnv-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSChangeEnv-env-op (alien-value to)) port env))
    (send to-STSChangeEnv send-maxwait send-option)))


;;; STSGrabWindow -- public
;;;
;;; User-side remote procedure call interface.
;;;
(defun STSGrabWindow (Remote_Port kPort)
  "The user side of remote procedure STSGrabWindow"
  (prog (R_e_s_u_l_t
         (send-waittime 0)
         (send-option 0)
         (receive-waittime 0)
         gr)
    (alien-bind ((to to-STSGrabWindow to-STSGrabWindow))
                (alien-store
                 (msg-remoteport-op (Msgize-to-STSGrabWindow-op (alien-value to)))
                 port Remote_Port)
                (alien-store (to-STSGrabWindow-kPort-op (alien-value to)) port
                             kPort)
                (setq gr
                      (TS-send (Msgize-to-STSGrabWindow-op (alien-value to))
                       to-STSGrabWindow send-waittime send-option)))
    (alien-bind ((from from-STSGrabWindow from-STSGrabWindow))
                (cond ((eql gr success)
                       (alien-store
                        (msg-msgsize-op
                         (Msgize-from-STSGrabWindow-op (alien-value from)))
                        (signed-byte 32) 36)
                       (alien-store
                        (msg-localport-op
                         (Msgize-from-STSGrabWindow-op (alien-value from)))
                        port *ReceivePort*)
                       (setq gr
                             (receive from-STSGrabWindow receive-waittime localpt
                                      receivelt))
                       (cond ((not (eql gr success))
                              NIL
                              (return gr))
                             ((=
                               (access-msg-id
                                (Msgize-from-STSGrabWindow-op (alien-value from)))
                               2912)
```

```
                        (cond ((=
                                (access-typetypei-typename
                                 (msg-retcode-typetypei-op
                                  (Msgize-from-STSGrabWindow-op
                                   (alien-value from))))
                                1)
                               (setq gr
                                     (access-msg-retcode
                                      (Msgize-from-STSGrabWindow-op
                                       (alien-value from)))))
                              (t
                               NIL
                               (return badreply)))
                        (cond ((eql
                                (access-typetypei-typename
                                 (from-STSGrabWindow-R_e_s_u_l_t-typetypeI-op
                                  (alien-value from)))
                                6)
                               (setq R_e_s_u_l_t
                                     (access-from-STSGrabWindow-R_e_s_u_l_t
                                      (alien-value from))))
                              (t
                               NIL
                               (return BADREPLY)))
                        NIL
                        (return (values R_e_s_u_l_t)))
                       (t
                        NIL
                        (return
                         (user-alternate-return
                          (Msgize-from-STSGrabWindow-op (alien-value from)))))))
                 (t
                  NIL
                  (return gr))))))


;;; STSPutCharArray -- public
;;;
;;; User-side message interface.
;;;
(defun STSPutCharArray (Remote_Port chars char_count firstCh lastCh)
  "The user side of message STSPutCharArray"
  (let ((send-maxwait 0)
        (send-option 0))
    (alien-bind ((to to-STSPutCharArray to-STSPutCharArray))
                (alien-store
                 (msg-remoteport-op
                  (Msgize-to-STSPutCharArray-op (alien-value to)))
                 port Remote_Port)
                (alien-store
                 (to-STSPutCharArray-chars-TypeNumObjects-Op (alien-value to))
                 (signed-byte 32) char_count)
                (alien-store (to-STSPutCharArray-chars-op (alien-value to))
                             system-area-pointer chars)
                (alien-store (to-STSPutCharArray-firstCh-op (alien-value to))
                             (signed-byte 16) firstCh)
                (alien-store (to-STSPutCharArray-lastCh-op (alien-value to))
                             (signed-byte 16) lastCh))
    (send to-STSPutCharArray send-maxwait send-option)))
```

```
;;; TSAsynch -- internal
;;;
;;; User-side asynchronous message dispatching function.
;;;
(defun TSAsynch (msg)
  "User-side asynchronous message dispatching function."
  (alien-bind ((alien-msg msg msg))
              (case (access-msg-id (alien-value alien-msg))
                (t badmsgid))))

;;; User-Alternate-Return -- internal
;;;
;;; User-side alternate return dispatching function.
;;;
(defun user-alternate-return (msg)
  "User-side alternate return dispatching function."
  (alien-bind ((alien-msg msg msg))
              (case (access-msg-id (alien-value alien-msg))
                (t badreply))))


(export
 '(TS-Init TSAsynch STSOpen STSOpenWindow STSFullOpen STSFullOpenWindow
  STSFullLine STSGetChar STSGetString STSPutChar STSPutString STSFlushInput
  STSFlushOutput STSChangeEnv STSGrabWindow STSPutCharArray))
```

11.2.6. TsMsgDefs.slisp

This file contains the Accent Lisp message definitions and is for compilation only.

```
;;;;   -*-Lisp-*-
;;;;
;;;;  Matchmaker generated msg definitions file for TS
;;;;
;;;;
;;;;            THIS FILE SHOULD NOT BE HAND EDITED
;;;;
;;;;  To change this interface, edit the interface matchmaker
;;;;  file and run it through matchmaker to generate this file.
;;;;


(in-package "TSDEFS")
(use-package "BUILTINDEFS")
(use-package "MMINTERNALDEFS")

(use-package "SAPPHDEFSDEFS")

(use-package "ACCINTDEFS")

(defoperator (Msgize-to-STSOpen-op msg) ((specific to-STSOpen)) '
             (alien-index (alien-value , specific) (bits 0) (bits 304)))

(defoperator (to-STSOpen-ize-op to-STSOpen) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 304)))


(defoperator (to-STSOpen-vp-op port) ((f to-STSOpen)) '
             (alien-index (alien-value , f) (bits 208) (bits 32)))

(defmacro Access-to-STSOpen-vp (r)
  '
  (alien-access (to-STSOpen-vp-op , r) port))

(defoperator (to-STSOpen-vp-TypeTypeI-op (signed-byte 32)) ((f to-STSOpen)) '
             (alien-index (alien-value , f) (bits 176) (bits 32)))


(defoperator (to-STSOpen-env-op port) ((f to-STSOpen)) '
             (alien-index (alien-value , f) (bits 272) (bits 32)))

(defmacro Access-to-STSOpen-env (r)
  '
  (alien-access (to-STSOpen-env-op , r) port))

(defoperator (to-STSOpen-env-TypeTypeI-op (signed-byte 32)) ((f to-STSOpen)) '
             (alien-index (alien-value , f) (bits 240) (bits 32)))

(defoperator (Msgize-from-STSOpen-op msg) ((specific from-STSOpen)) '
             (alien-index (alien-value , specific) (bits 0) (bits 288)))

(defoperator (from-STSOpen-ize-op from-STSOpen) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 288)))


(defoperator (from-STSOpen-R_e_s_u_l_t-op port) ((f from-STSOpen)) '
             (alien-index (alien-value , f) (bits 256) (bits 32)))

(defmacro Access-from-STSOpen-R_e_s_u_l_t (r)
  '
  (alien-access (from-STSOpen-R_e_s_u_l_t-op , r) port))
```

```
(defoperator (from-STSOpen-R_e_s_u_l_t-TypeTypeI-op (signed-byte 32))
             ((f from-STSOpen)) '
             (alien-index (alien-value , f) (bits 224) (bits 32)))

(defoperator (Msgize-to-STSOpenWindow-op msg) ((specific to-STSOpenWindow)) '
             (alien-index (alien-value , specific) (bits 0) (bits 304)))

(defoperator (to-STSOpenWindow-ize-op to-STSOpenWindow) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 304)))


(defoperator (to-STSOpenWindow-w-op port) ((f to-STSOpenWindow)) '
             (alien-index (alien-value , f) (bits 208) (bits 32)))

(defmacro Access-to-STSOpenWindow-w (r)
  '
  (alien-access (to-STSOpenWindow-w-op , r) port))

(defoperator (to-STSOpenWindow-w-TypeTypeI-op (signed-byte 32))
             ((f to-STSOpenWindow)) '
             (alien-index (alien-value , f) (bits 176) (bits 32)))


(defoperator (to-STSOpenWindow-env-op port) ((f to-STSOpenWindow)) '
             (alien-index (alien-value , f) (bits 272) (bits 32)))

(defmacro Access-to-STSOpenWindow-env (r)
  '
  (alien-access (to-STSOpenWindow-env-op , r) port))

(defoperator (to-STSOpenWindow-env-TypeTypeI-op (signed-byte 32))
             ((f to-STSOpenWindow)) '
             (alien-index (alien-value , f) (bits 240) (bits 32)))

(defoperator (Msgize-from-STSOpenWindow-op msg) ((specific from-STSOpenWindow)) '
             (alien-index (alien-value , specific) (bits 0) (bits 288)))

(defoperator (from-STSOpenWindow-ize-op from-STSOpenWindow) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 288)))


(defoperator (from-STSOpenWindow-R_e_s_u_l_t-op port) ((f from-STSOpenWindow)) '
             (alien-index (alien-value , f) (bits 256) (bits 32)))

(defmacro Access-from-STSOpenWindow-R_e_s_u_l_t (r)
  '
  (alien-access (from-STSOpenWindow-R_e_s_u_l_t-op , r) port))
```

```
(defoperator (from-STSOpenWindow-R_e_s_u_l_t-TypeTypeI-op (signed-byte 32))
             ((f from-STSOpenWindow)) '
             (alien-index (alien-value . f) (bits 224) (bits 32)))

(defoperator (Msgize-to-STSFullOpen-op msg) ((specific to-STSFullOpen)) '
             (alien-index (alien-value . specific) (bits 0) (bits 2480)))

(defoperator (to-STSFullOpen-ize-op to-STSFullOpen) ((generic msg)) '
             (alien-index (alien-value . generic) (bits 0) (bits 2480)))



(defoperator (to-STSFullOpen-vp-op port) ((f to-STSFullOpen)) '
             (alien-index (alien-value . f) (bits 208) (bits 32)))

(defmacro Access-to-STSFullOpen-vp (r)
  '
  (alien-access (to-STSFullOpen-vp-op . r) port))

(defoperator (to-STSFullOpen-vp-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpen)) '
             (alien-index (alien-value . f) (bits 176) (bits 32)))


(defoperator (to-STSFullOpen-env-op port) ((f to-STSFullOpen)) '
             (alien-index (alien-value . f) (bits 272) (bits 32)))

(defmacro Access-to-STSFullOpen-env (r)
  '
  (alien-access (to-STSFullOpen-env-op . r) port))

(defoperator (to-STSFullOpen-env-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpen)) '
             (alien-index (alien-value . f) (bits 240) (bits 32)))


(defoperator (to-STSFullOpen-fontName-op (perq-string 255)) ((f to-STSFullOpen))
             ' (alien-index (alien-value . f) (bits 336) (bits 2048)))

(defmacro Access-to-STSFullOpen-fontName (r)
  '
  (alien-access (to-STSFullOpen-fontName-op . r) simple-string))

(defoperator (to-STSFullOpen-fontName-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpen)) '
             (alien-index (alien-value . f) (bits 304) (bits 32)))


(defoperator (to-STSFullOpen-doWrap-op boolean) ((f to-STSFullOpen)) '
             (alien-index (alien-value . f) (bits 2416) (bits 1)))
```

```
(defmacro Access-to-STSFullOpen-doWrap (r)
  '
  (alien-access (to-STSFullOpen-doWrap-op , r) boolean))

(defoperator (to-STSFullOpen-doWrap-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpen)) '
             (alien-index (alien-value , f) (bits 2384) (bits 32)))


(defoperator (to-STSFullOpen-dispPages-op (signed-byte 16)) ((f to-STSFullOpen))
             ' (alien-index (alien-value , f) (bits 2464) (bits 16)))

(defmacro Access-to-STSFullOpen-dispPages (r)
  '
  (alien-access (to-STSFullOpen-dispPages-op , r) (signed-byte 16)))

(defoperator (to-STSFullOpen-dispPages-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpen)) '
             (alien-index (alien-value , f) (bits 2432) (bits 32)))

(defoperator (Msgize-from-STSFullOpen-op msg) ((specific from-STSFullOpen)) '
             (alien-index (alien-value , specific) (bits 0) (bits 288)))

(defoperator (from-STSFullOpen-ize-op from-STSFullOpen) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 288)))



(defoperator (from-STSFullOpen-R_e_s_u_l_t-op port) ((f from-STSFullOpen)) '
             (alien-index (alien-value , f) (bits 256) (bits 32)))

(defmacro Access-from-STSFullOpen-R_e_s_u_l_t (r)
  '
  (alien-access (from-STSFullOpen-R_e_s_u_l_t-op , r) port))

(defoperator (from-STSFullOpen-R_e_s_u_l_t-TypeTypeI-op (signed-byte 32))
             ((f from-STSFullOpen)) '
             (alien-index (alien-value , f) (bits 224) (bits 32)))

(defoperator (Msgize-to-STSFullOpenWindow-op msg)
             ((specific to-STSFullOpenWindow)) '
             (alien-index (alien-value , specific) (bits 0) (bits 2480)))

(defoperator (to-STSFullOpenWindow-ize-op to-STSFullOpenWindow) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 2480)))



(defoperator (to-STSFullOpenWindow-w-op port) ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 208) (bits 32)))

(defmacro Access-to-STSFullOpenWindow-w (r)
  '
  (alien-access (to-STSFullOpenWindow-w-op , r) port))

(defoperator (to-STSFullOpenWindow-w-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 176) (bits 32)))


(defoperator (to-STSFullOpenWindow-env-op port) ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 272) (bits 32)))

(defmacro Access-to-STSFullOpenWindow-env (r)
  '
  (alien-access (to-STSFullOpenWindow-env-op , r) port))
```

```
(defoperator (to-STSFullOpenWindow-env-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 240) (bits 32)))


(defoperator (to-STSFullOpenWindow-fontName-op (perq-string 256))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 336) (bits 2048)))

(defmacro Access-to-STSFullOpenWindow-fontName (r)
  '
  (alien-access (to-STSFullOpenWindow-fontName-op , r) simple-string))

(defoperator (to-STSFullOpenWindow-fontName-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 304) (bits 32)))


(defoperator (to-STSFullOpenWindow-doWrap-op boolean) ((f to-STSFullOpenWindow))
             ' (alien-index (alien-value , f) (bits 2416) (bits 1)))

(defmacro Access-to-STSFullOpenWindow-doWrap (r)
  '
  (alien-access (to-STSFullOpenWindow-doWrap-op , r) boolean))

(defoperator (to-STSFullOpenWindow-doWrap-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 2384) (bits 32)))


(defoperator (to-STSFullOpenWindow-dispPages-op (signed-byte 16))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 2464) (bits 16)))
```

```
(defmacro Access-to-STSFullOpenWindow-dispPages (r)
  '
  (alien-access (to-STSFullOpenWindow-dispPages-op , r) (signed-byte 16)))

(defoperator (to-STSFullOpenWindow-dispPages-TypeTypeI-op (signed-byte 32))
             ((f to-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 2432) (bits 32)))

(defoperator (Msgize-from-STSFullOpenWindow-op msg)
             ((specific from-STSFullOpenWindow)) '
             (alien-index (alien-value , specific) (bits 0) (bits 288)))

(defoperator (from-STSFullOpenWindow-ize-op from-STSFullOpenWindow)
             ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 288)))


(defoperator (from-STSFullOpenWindow-R_e_s_u_l_t-op port)
             ((f from-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 256) (bits 32)))

(defmacro Access-from-STSFullOpenWindow-R_e_s_u_l_t (r)
  '
  (alien-access (from-STSFullOpenWindow-R_e_s_u_l_t-op , r) port))

(defoperator (from-STSFullOpenWindow-R_e_s_u_l_t-TypeTypeI-op (signed-byte 32))
             ((f from-STSFullOpenWindow)) '
             (alien-index (alien-value , f) (bits 224) (bits 32)))

(defoperator (Msgize-to-STSFullLine-op msg) ((specific to-STSFullLine)) '
             (alien-index (alien-value , specific) (bits 0) (bits 176)))
```

```
(defoperator (to-STSFullLine-ize-op to-STSFullLine) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 176)))


(defoperator (Msgize-from-STSFullLine-op msg) ((specific from-STSFullLine)) '
             (alien-index (alien-value , specific) (bits 0) (bits 272)))

(defoperator (from-STSFullLine-ize-op from-STSFullLine) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 272)))



(defoperator (from-STSFullLine-R_e_s_u_l_t-op boolean) ((f from-STSFullLine)) '
             (alien-index (alien-value , f) (bits 256) (bits 1)))

(defmacro Access-from-STSFullLine-R_e_s_u_l_t (r)
  '
  (alien-access (from-STSFullLine-R_e_s_u_l_t-op , r) boolean))

(defoperator (from-STSFullLine-R_e_s_u_l_t-TypeTypeI-op (signed-byte 32))
             ((f from-STSFullLine)) '
             (alien-index (alien-value , f) (bits 224) (bits 32)))

(defoperator (Msgize-to-STSGetChar-op msg) ((specific to-STSGetChar)) '
             (alien-index (alien-value , specific) (bits 0) (bits 176)))

(defoperator (to-STSGetChar-ize-op to-STSGetChar) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 176)))


(defoperator (Msgize-from-STSGetChar-op msg) ((specific from-STSGetChar)) '
             (alien-index (alien-value , specific) (bits 0) (bits 272)))

(defoperator (from-STSGetChar-ize-op from-STSGetChar) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 272)))



(defoperator (from-STSGetChar-R_e_s_u_l_t-op string-char) ((f from-STSGetChar)) '
             (alien-index (alien-value , f) (bits 256) (bits 8)))

(defmacro Access-from-STSGetChar-R_e_s_u_l_t (r)
  '
  (alien-access (from-STSGetChar-R_e_s_u_l_t-op , r) string-char))

(defoperator (from-STSGetChar-R_e_s_u_l_t-TypeTypeI-op (signed-byte 32))
             ((f from-STSGetChar)) '
             (alien-index (alien-value , f) (bits 224) (bits 32)))

(defoperator (Msgize-to-STSGetString-op msg) ((specific to-STSGetString)) '
             (alien-index (alien-value , specific) (bits 0) (bits 176)))

(defoperator (to-STSGetString-ize-op to-STSGetString) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 176)))


(defoperator (Msgize-from-STSGetString-op msg) ((specific from-STSGetString)) '
             (alien-index (alien-value , specific) (bits 0) (bits 2304)))

(defoperator (from-STSGetString-ize-op from-STSGetString) ((generic msg)) '
             (alien-index (alien-value , generic) (bits 0) (bits 2304)))
```