TTSMAN
Versione per Java e .Net

16.5.2014
Vers. 2.8

INDICE:

⇐ INDICE

TTSMAN in Java e .NET

Il TTSMAN permette di sviluppare in ambiente Java e .NET le applicazioni di raccolta dati con il sistema TTS.

Le classi principali sono:

- Ttsman
- Configuratore_Sinottico

La classe Ttsman gestisce la comunicazione con l'hardware TTS e permette di ricevere e trasmettere le sue stringhe nell'applicazione di raccolta dati.

La classe Configuratore_Sinottico, con la sua interfaccia grafica, permette di generare i file "confObjectTTS.xml" e(o) "ttsparm.txt", che contengono le stringhe di configurazione di tutti i dispositivi TTS presenti; un file viene utilizzato da un oggetto Ttsman per inviare le stringhe di configurazione ai vari dispositivi.
Il file "confObjectTTS.xml" può essere sostituito da un Database.

Per trasmettere una stringa ad un dispositivo TTS (RS 232C, display, digital I/O, ecc.), il driver TTSMAN prevede la seguente struttura gerarchica delle classi (illustrate più avanti):

- Ttsman,
- Master,
- Slave,
- Hwobject;

ad esempio per inviare la stringa "Prova Sistema TTS" al display di uno slave l'istruzione è:

test. IPTTS[i_mast].TTS[i_slv].display.s_dat ("Prova sistema TTS");

dove:

- test è un oggetto di classe Ttsman;
- IPTTS [i_mast] è un elemento dell'array IPTTS di classe Master, i_mast è l'indice (da o a 254) del master TTS;
- TTS[i_slv] è un elemento dell'array TTS di classe Slave, i_slv è l'indice (da 1 a 254) dello slave TTS;
- display è un oggetto di classe Hwobject (dispositivo appartenente ad uno slave TTS);
- s_dat è il metodo della classe Hwobject, che costruisce la stringa dati e la trasmette tramite il socket dell'oggetto IPTTS[i_mast].

La ricezione di una stringa avviene tramite i metodi della classe StrTTS.

L'applicazione deve:

- prevedere una classe che estenda StrTTS,a tale scopo e' fornito un modello(TTSImplModel) e degli esempi;di seguito ne è riportato uno.

```
class testGeneric extends StrTTS
```

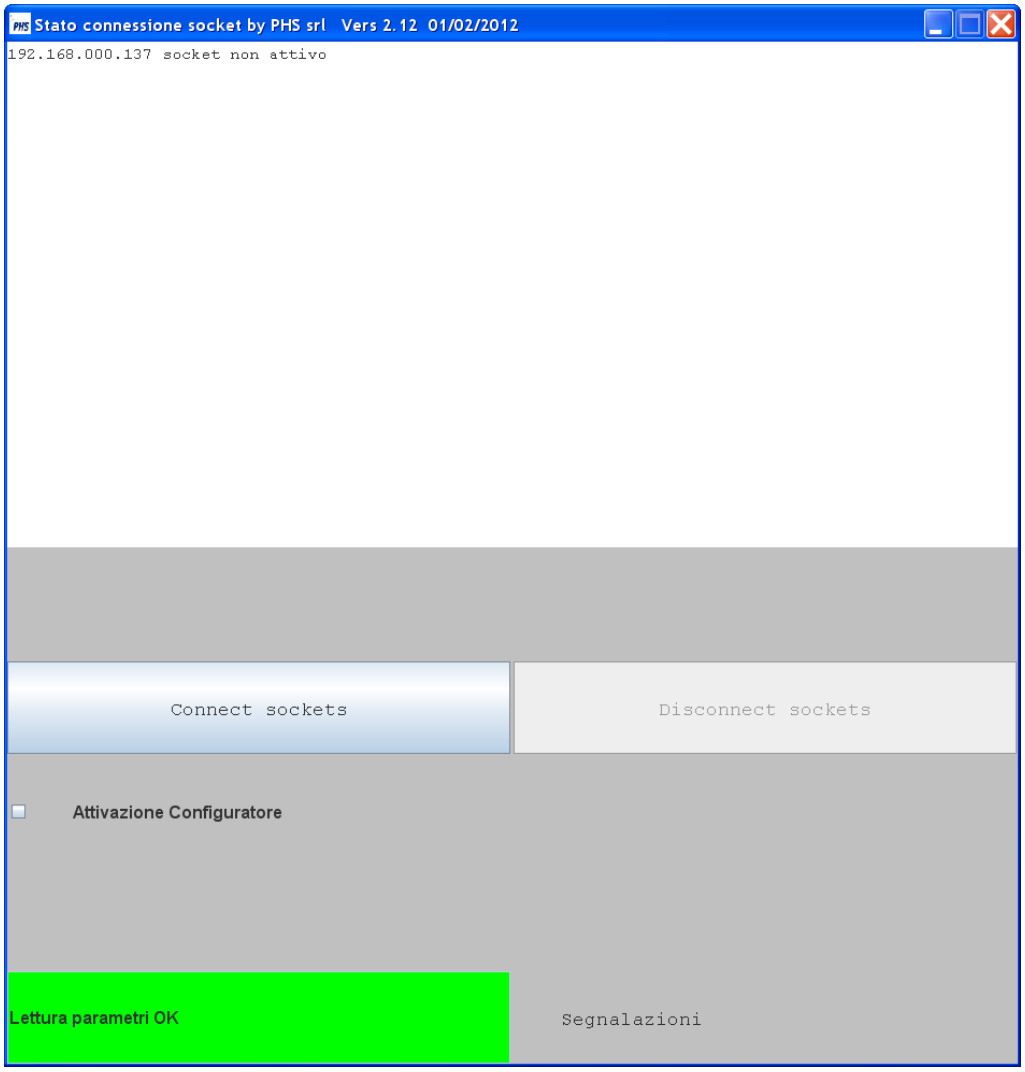
- creare un oggetto di classe Ttsman,ad esempio:

```
testGeneric testGen = new testGeneric();

Ttsman test = testGen.getTtsManInst();
```

Alla partenza di un'applicazione è presentato il pannello seguente se:

```
setFl_panc(true);
setFl_synopt(false);
```



il pannello ha lo scopo di fornire un modo rapido per utilizzare dei metodi di classe Ttsman:

- Connect sockets : usa il metodo actAllMasters();

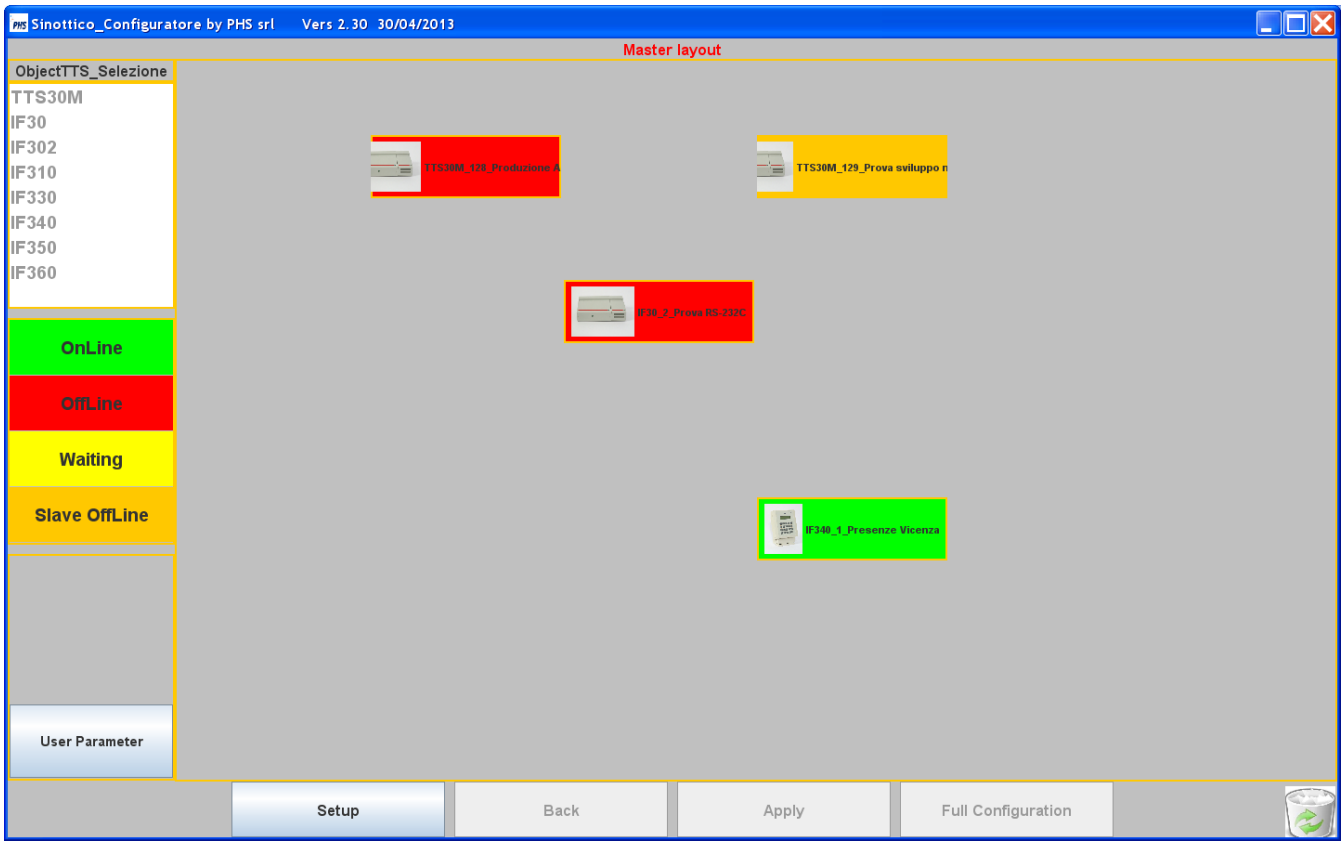
- Disconnect sockets : usa il metodo deactAllMasters();

- Attivazione Configuratore : usa il metodo startConfig().

⇐ INDICE

Si può decidere di utilizzare il pannello sinottico se :

```
setFl_panc(false);
setFl_synopt(true);
```



Il pannello (descritto nel documento “TTSSDK.pdf”),permette di configurare i vari dispositivi e di visualizzarne lo stato.

E’ possibile utilizzare due strutture applicative:

-applicazione integrata,

-applicazione e configuratore_sinottico disgiunti.

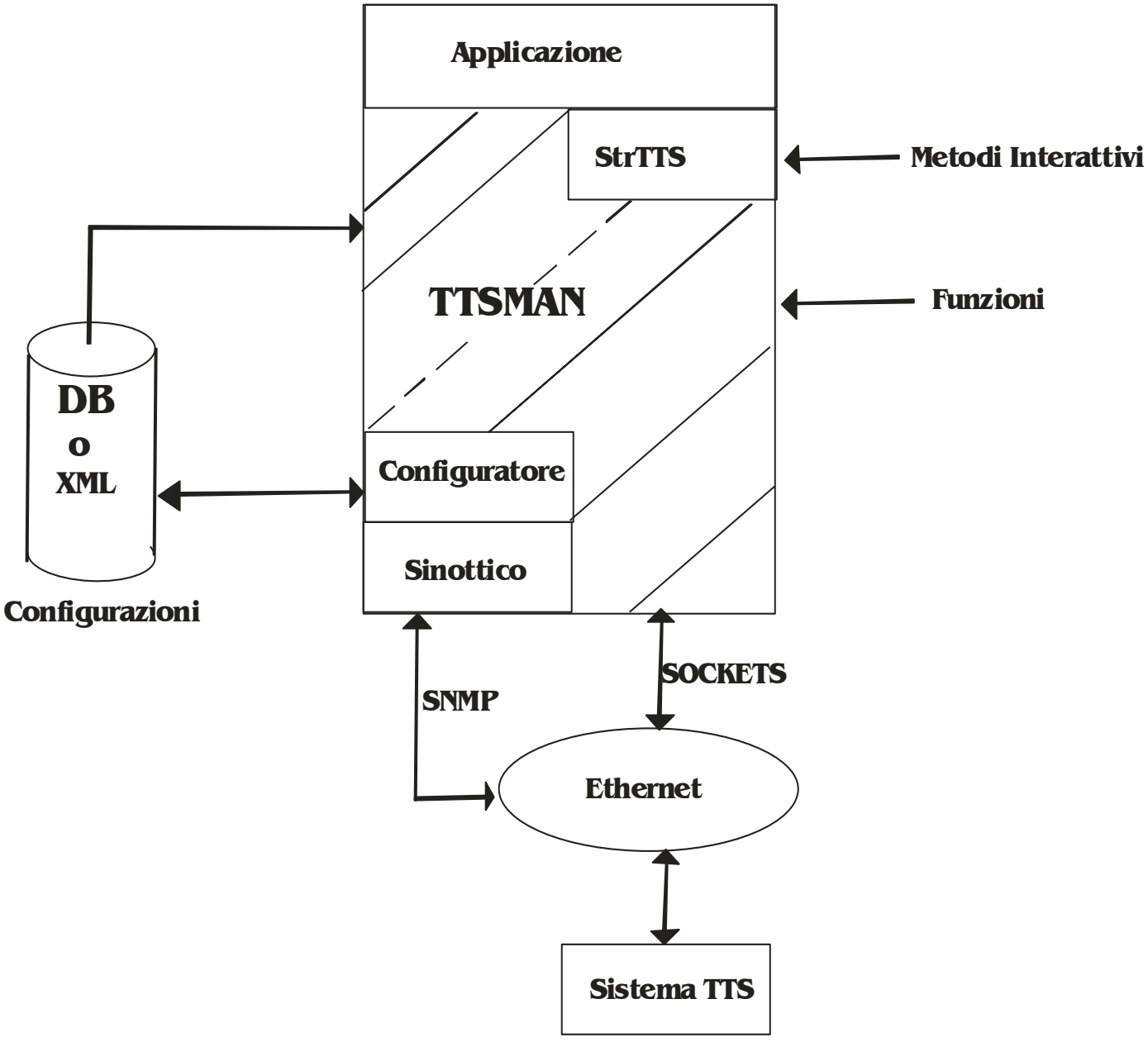
⇐ INDICE

Applicazione Integrata

Lo schema seguente rappresenta l'architettura di un'applicazione,in cui il codice ed i form appartengono entrambi ad essa,e che utilizza i metodi della classe StrTTS e le funzioni di quella Ttsman.

La parametrizzazione da usare è:

```
setFl_panc(false);
setFl_synopt(true);
setFl_extpar(true);
```



⇐ INDICE

Il Configuratore gestisce il file “confObjectTTS.xml” o accede ad un Database (come descritto più avanti.

Il TTSMAN (oggetto di classe Ttsman) legge il file (o accede al Database) e configura tutti i dispositi previsti ed attivi.

Il Sinottico visualizza lo stato dei terminali,mediante variabili fornite dal TTSMAN e,se possibile, tramite SNMP.

La classe StrTTS fornisce l’oggetto “listType”,che permette di escludere i dispositivi che non debbono esser gestiti;ad esempio:

```
listType.tts30M = false;
```

la configurazione di default è:

```
listType.tts30M   = true;
listType.if30     = true;
listType.if302    = true;
listType.if310    = true;
listType.if330    = true;
listType.if340    = true;
listType.tif350   = true;
listType.if360    = true;
listType.machine = false;
listType.hostcon  = false;
```

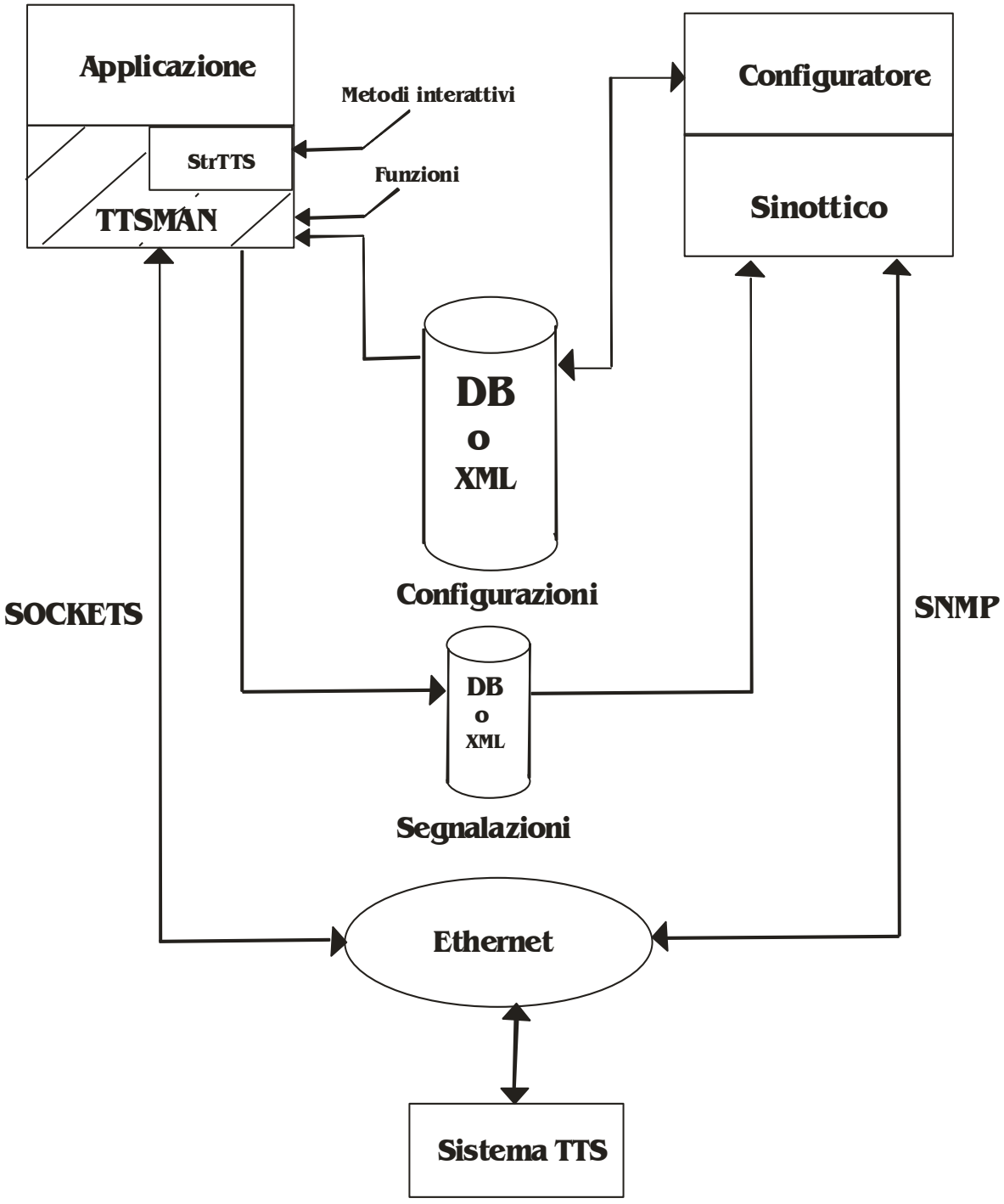
⇐ INDICE

Applicazione e Configuratore disgiunti

Lo schema seguente rappresenta l'architettura in cui sono presenti due applicazioni.

La prima ha solo codice e nessun form,essa utilizza i metodi della classe StrTTS e le funzioni di quella Ttsman.

L'altra ha tutti i form ed il codice minimo per estendere la classe StrTTS.



La parametrizzazione da usare,per la prima applicazione,è:

```
setFl_panc(false);
setFl_synopt(false);
setFl_extpar(true);
```

per la seconda:

```
setFl_panc(false);
setFl_synopt(true);
setFl_extpar(true);
getCommState().setExternMode(true);
getCommState().create();
```

Il Configuratore gestisce il file “confObjectTTS.xml” o accede ad un Database.

Il TTSMAN(oggetto di classe Ttsman) legge il file “confObjectTTS.xml” (o accede al Database) e configura tutti i dispositi previsti ed attivi; inoltre genera il file “stateObjectTTS.xml”,riportante lo stato dei dispositivi previsti;come il file “signalObjectTTS.xml” per lo stato di attività e vitalità dell’applicazione a cui appartiene.

Il Sinottico visualizza lo stato dei terminali e dell’applicazione,leggendo i file “stateObjectTTS.xml” e “signalObjectTTS.xml” e,se possibile,tramite SNMP.

La classe StrTTS fornisce l’oggetto “listType”,che permette di escludere i dispositivi che non debbono esser gestiti nell’applicazione Configuratore_Sinottico;ad esempio:

```
listType.tts30M = false;
```

la configurazione di default è:

```
listType.tts30M   = true;
listType.if30     = true;
listType.if302    = true;
listType.if310    = true;
listType.if330    = true;
listType.if340    = true;
listType.tif350   = true;
listType.if360    = true;
listType.machine = false;
listType.hostcon  = false;
```

L’architettura illustrata permette di sviluppare una tantum il Configuratore_Sinottico come client grafico ed di volta in volta un’applicazione server di raccolta dati attivabile anche come servizio windows, in questo caso il file “signalObjectTTS.xml” deve riportarne l’attivazione e la disattivazione.

⇐ INDICE

Configurazioni e segnalazioni su Database

Il file “confObjectTTS.xml” può essere sostituito con un Database tramite l’override dei metodi della classe StrTTS:

getAllConf(),

setAllConf().

Il sinottico,invece,può essere alimentato da un Database con l’override dei metodi (richiedenti lo stato di un dispositivo) della classe StrTTS:

synMasterState(),

synSlaveState(),

synMachineState(),

synConnHost().

⇐ INDICE

Recupero ttsparm.txt

E’possibile recuperare il file “ttsparm.txt” convertendolo nel “confObjectTTS.xml” o riportandolo in un Database.

Bisogna partire con la configurazione:

```
setFl_panc(false);
setFl_synopt(true);
setFl_extpar(false);
```

e poi ritornare nella configurazione:

```
setFl_panc(false);
setFl_synopt(true);
setFl_extpar(true);
```

Se il file “ttsparm.txt” è presente viene aggiornato come backup delle configurazioni.

⇐ INDICE

Esempio di utilizzo della classe StrTTS

```
package testUser;

import java.io.File;
import java.util.ArrayList;
import ttsMAN.Master;
import ttsMAN.Strframe;
import ttsMAN.Ttsman;
import userTTS.*;

public class testGeneric extends StrTTS
{
		public testGeneric()
		{
			super();

			setFl_panc(true);		//abilitazione pannello stato connessione socket
			setFl_synopt(false);		//esclusione pannello sinottico
			setFl_extpar(false);		//utilizzo file ttsparm.txt
			setFl_consol(true);		//abilitazione console

			//Copia del file di configurazione
			setVConfigFile(new File("ttsparm_generic.txt"));

			initialize();
		}

		public static void main(String[] args)
		{
			//Oggetto per rendere disponibili i metodi per la gestione degli oggetti HW
			testGeneric testGen = new testGeneric();

			Ttsman test = testGen.getTtsManInst();

			try
			{
				int i_mast = 2;
				int i_slv = 2;

				do
				{
					if(test.getStateMaster( i_mast) == 0 && test.getStateSlave(i_mast,i_slv) == 0)
					{
						try
						{
							//ArrayList temp = test.getStructULSlave(2, 2);

							//test.IPTTS[i_mast].TTS[i_slv].display.s_dat("Ciao Giuseppone");
							//test.IPTTS[i_mast].TTS[i_slv].buzzer.s_dat("30?0?");
							//test.IPTTS[i_mast].TTS[i_slv].rs232c.s_cmd("0");

							//Polling SCIROCCO
							//String gbc = "80910101";
							//test.IPTTS[i_mast].TTS[i_slv].rs232c.s_dat(gbc);

							//Polling PLC FANUC
							//String gbc = ":00103000C0002";
							//gbc = ":0ABCDEF";
							//test.IPTTS[i_mast].TTS[i_slv].rs232c.s_cmd(gbc);
							//gbc = ":012345678";
							//test.IPTTS[i_mast].TTS[i_slv].rs232c.s_cmd(gbc);
						}
						catch (Exception e){e.printStackTrace();}
					}
					Thread.currentThread().yield();
					Thread.currentThread().sleep(5000);

				} while (true);
			} catch (Exception e) {		e.printStackTrace();}
		}
```

```
/*
	******
****** Metodi per la gestione degli oggetti HW TTS **********    *

	******
*/

public void VSC(Master IPTTS, String adVSC, Strframe letsocket)
{
}
public void ADCONV(Master IPTTS, int isl, int ish, Strframe letsocket)
{
	try
	{
		IPTTS.TTS[isl].display.s_dat("Prova A/Dconv   ");
		IPTTS.TTS[isl].buzzer.s_dat("10?");
	}
	catch (Exception e){e.printStackTrace();}
}

public void BADGE(Master IPTTS, int isl, int ish, Strframe letsocket,
		String[] CmaskF)
{
	try
	{
		IPTTS.TTS[isl].display.s_cmd("3");
		IPTTS.TTS[isl].display.s_dat("Prova Badge     ");
		IPTTS.TTS[isl].buzzer.s_dat("20?");
	}
	catch (Exception e){e.printStackTrace();}
}

public void BARCOD(Master IPTTS, int isl, int ish, Strframe letsocket)
{
	try
	{
		IPTTS.TTS[isl].display.s_cmd("3");
		IPTTS.TTS[isl].display.s_dat("Prova Barcode   ");
		IPTTS.TTS[isl].buzzer.s_dat("20?");
	}
	catch (Exception e){e.printStackTrace();}
}

public void CLK(Master IPTTS, int isl, int ish, Strframe letsocket)
{
	try
	{
		IPTTS.TTS[isl].display.s_cmd("3");
		IPTTS.TTS[isl].display.s_dat("Prova Orologio  ");
		IPTTS.TTS[isl].buzzer.s_dat("10?");
	}
	catch (Exception e){e.printStackTrace();}
}

public void DIGIO(Master IPTTS, int isl, int ish, Strframe letsocket)
{
	int adCount = 0;
	int counti[] = new int[8];
	for (int i = 0; i < 8; i++) { counti[i] = 0; }
	String stPort[] = new String[16];
	for (int i = 0; i < 16; i++) { stPort[i] = ""; }

	switch (letsocket.Frame.substring(4).length())
	{
		case 1:
		{
			//Stato porta 2 bit
			try
			{
				//IPTTS.TTS[isl].orologio.s_wconf("02291107173005");
			}
			catch (Exception e){e.printStackTrace();}
		}break;
		case 2:
		{
			//Inattivita' contapezzi
```

```
                        adCount = Integer.parseInt(letsocket.Frame.substring(4, 5));
                } break;
                case 4:
                {
                        //Stato porta 16 bit M = 0
                } break;

                case 16:
                {
                        //Stato porta 16 bit M = 1
                        for (int i=0;i<16;i++)
                        {
                                stPort[i] = letsocket.Frame.substring(4 + i, 5 + i);
                        }
                } break;
                default:
                {
                        int nCount = Integer.parseInt(letsocket.Frame.substring(4)) / 5;

                        // Estrazione contatori
                        for (int i=0;i<nCount;i++)
                        {
                                adCount = Integer.parseInt(letsocket.Frame.substring(4+i*5, 5+i*5));
                                counti[adCount] = Integer.parseInt(letsocket.Frame.substring(5 + i * 5, 9 + i * 5));
                        }
                }
        }

        try
        {
                IPTTS.TTS[isl].display.s_cmd("3");
                IPTTS.TTS[isl].display.s_dat("Prova Digital   ");
                IPTTS.TTS[isl].buzzer.s_dat("10?");
                IPTTS.TTS[isl].rele.s_dat("10?");
        }
        catch (Exception e){e.printStackTrace();}
}

public void GLOBAC(Master IPTTS, int isl, int ish, Strframe letsocket,
                String[] CmaskF)
{
}

public void GLOBAD(Master IPTTS, int isl, int ish, Strframe letsocket,
                String[] CmaskF)
{
        try
        {
                if (letsocket.Frame.substring(4).equals("07"))
                {
                        //Clear buffer
                        IPTTS.TTS[isl].global.s_cmd("4");

                        IPTTS.TTS[isl].display.s_dat(CmaskF[1]);
                }
        }
        catch (Exception e){e.printStackTrace();}
}

public void GLOBAL(Master IPTTS, int isl, int ish, Strframe letsocket)
{
        try
        {
                IPTTS.TTS[isl].display.s_cmd("3");
                IPTTS.TTS[isl].display.s_dat("Prova Global     ");
                IPTTS.TTS[isl].buzzer.s_dat("10?");
        }
        catch (Exception e){e.printStackTrace();}
}

public void KEYB(Master IPTTS, int isl, int ish, Strframe letsocket,
                String[] CmaskF)
{
        char SO = 14;

        try
        {
```

```
                //Tasti funzione
                if (letsocket.Frame.substring(4).equals("&1"))
                {
                        //F1
                        IPTTS.TTS[isl].rs232c.s_cmd("0");
                        IPTTS.TTS[isl].rele.s_dat("11");
                        IPTTS.TTS[isl].display.s_cmd("3");
                        IPTTS.TTS[isl].display.s_dat("Prova Tastiera  by PHS");
                        IPTTS.TTS[isl].global.s_cmd("0");
                }
                else if (letsocket.Frame.substring(4).equals("&2"))
                {
                        //F2
                        IPTTS.TTS[isl].display.s_dat(CmaskF[0]);
                }
                else if (letsocket.Frame.substring(4).equals("&3"))
                {
                        //F3
                        IPTTS.TTS[isl].display.s_dat(CmaskF[1]);
                }
                else if (letsocket.Frame.substring(4).equals("&4"))
                {
                        //F4
                }
                else
                {
                        //Videate
                        if (letsocket.Frame.substring(4,5).equals("" + SO))
                        {
                                //Display formattato
                                //Struttura input/object
                                ArrayList structdisplay;
                                structdisplay = ExstactDisplay(letsocket.Frame.substring(4));

                                for (int i=0;i<structdisplay.size();i=i+2)
                                {
                                        //Estrazione input/object
                                        String inputdis = structdisplay.get(i).toString();
                                        String objdis = structdisplay.get(i+1).toString();

                                        //Riconoscimento videata per object
                                        if (objdis.equals("msk1"))
                                        {
                                                IPTTS.TTS[isl].display.s_dat(CmaskF[1]);
                                        }
                                        else if (objdis.equals("msk2"))
                                        {
                                                IPTTS.TTS[isl].display.s_dat(CmaskF[0]);
                                        }
                                        else
                                        {
                                                //IPTTS.TTS[isl].display.s_dat(CmaskF[0]);
                                        }
                                }
                        }
                        else
                        {
                                //Formato libero
                                IPTTS.TTS[isl].display.s_cmd("3");
                                IPTTS.TTS[isl].display.s_dat("Prova Tastiera  by PHS");
                                IPTTS.TTS[isl].global.s_cmd("0");
                        }
                }

                IPTTS.TTS[isl].buzzer.s_dat("10?");
                //IPTTS.TTS[isl].rele.s_dat("50?");
        }
        catch (Exception e){e.printStackTrace();}
}

public void RS232C(Master IPTTS, int isl, int ish, Strframe letsocket)
{
        try
        {
                IPTTS.TTS[isl].display.s_cmd("3");
                IPTTS.TTS[isl].display.s_dat("Code : " + letsocket.Frame.substring(5));
```

```
                        //IPTTS.TTS[isl].buzzer.s_dat("10?");
                        //IPTTS.TTS[isl].rs232c.s_cmd("0");

                        //Prova SCIROCCO
                        if (!letsocket.Frame.substring(4,10).equals("80F007") &&
                                        !letsocket.Frame.substring(4,10).equals("80F000"))
                        {
                                //String clr = "80920101";
                                //IPTTS.TTS[isl].rs232c.s_dat(clr);
                        }

                }
                catch (Exception e){e.printStackTrace();}
        }

        public void RS232D(Master IPTTS, int isl, int ish, Strframe letsocket)
        {
                try
                {
                        IPTTS.TTS[isl].rs232c.s_dat("Sistema TTS");
                }
                catch (Exception e){e.printStackTrace();}
        }

        public void VAC(Master IPTTS, String adVAC, Strframe letsocket)
        {
                int ivac = Integer.parseInt(adVAC);

                try
                {
                        IPTTS.VAC[ivac].s_dat(adVAC,"Prova VAC");
                }
                catch (Exception e){e.printStackTrace();}
        }

        public void isSlaveOnline(Master IPTTS, int isl,boolean conn)
        {
                if (conn){System.out.println(IPTTS.Amast.toString() + " Slave " + isl + " online ");}
                else{System.out.println(IPTTS.Amast.toString() + " Slave " + isl + " offline ");}
        }

        public void isMasterOnline(Master IPTTS,boolean conn)
        {
                if (conn){System.out.println(IPTTS.Amast.toString()+ " Master" + " online ");}
                else{System.out.println(IPTTS.Amast.toString() + " Master" + " offline ");}
        }

        public void endConfiguration(Master IPTTS, int isl)
        {
                System.out.println(IPTTS.Amast.toString() + " Slave " + isl + " configurato ");
        }
}
```

⇐ INDICE

Class StrTTS

public class **StrTTS**

La classe permette,in particolare,di ricevere le stringhe generate dal sistema TTS, rendendo disponibili dei metodi che si attivano a seguito dell'evento relativo.

La classe e' una superclasse,che deve essere estesa.

Field Summary	
private boolean	**fl_ synopt** la variabile abilita o disabilita il pannello Sinottico_Configuratore.
private boolean	**fl_ extpar** la variabile abilita o disabilita, per i parametri del sistema TTS, l'uso di un DB o del file "confObjectTTS.xml" invece del file "ttsparm.txt".
private boolean	**fl_ panc** la variabile abilita o disabilita, in alternativa al pannello Sinottico_Configuratore, l'uso di uno altro semplificato.
private boolean	**fl_ consol** la variabile abilita o disabilita la consolle.
private Ttsman	**ttsManInst** oggetto di classe Tttsman.
private CommState	**commState** oggetto di classe CommState.
public StructImiDB	**structIniDB** oggetto di classe StructIniDB.
public StructImiVAC	**structIniVAC** oggetto di classe StructIniVAC.
public ArrayList	**parmUser** gli elementi dell'Arraylist sono del tipo <ArrayList<FieldColumn>>, questo permette di gestire, in configurazione, dei parametri generici di tipo FieldColumn, tramite i metodi getParmUser e setParmUser.

⇐ INDICE

Method Summary	
void	**Initialize** () Il metodo crea un oggetto di classe Ttsman,con riferimento alle variabili di instanza fl_panc,fl_consol,fl_extpar,fl_synopt.
Ttsman	**getTtsManInst** () Il metodo restituisce l'oggetto di classe Ttsman creato da initialize().
void	**setFl_panc** (boolean fl_panc) Il metodo assegna la variabile fl_panc.
void	**setFl_consol** (boolean fl_consol) Il metodo assegna la variabile fl_consol.
void	**setFL_synopt** (boolean fl_synopt) Il metodo assegna la variabile fl_synopt.
CommState	**getCommState** () Il metodo restituisce l'oggetto di classe CommState creato da initialize().
void	**setVConfigFile** (File configFile) Il metodo definisce il file di configurazione da sovrascrivere al file "ttsparm.txt".
void	**RS232C** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF) Evento dati generato dalla porta seriale RS 232C COM1.
void	**VAC** (Master IPTTS,String adVAC, Strframe letsocket) Evento dati generato dal canale VAC di indirizzo adVAC.

void	**VSC** (Master IPTTS,String adVSC, Strframe letsocket) Evento dati generato dal canale VSC di indirizzo adVSC.
void	**MACHINE** (Machine mach, Strframe letsocket) Evento dati generato da una macchina di tipo Machine.
boolean	**parseMachine** (ConnMachine machParm, Object file) Evento generato da una macchina di tipo Machine,per permettere il parsing o il filtro di un campo.
void	**BARCOD** (Master IPTTS,int isl,int ish , Strframe letsocket) Evento dati generato dalla porta barcode.
void	**BADGE** (Master IPTTS,int isl,int ish , Strframe letsocket,String[] CmaskF) Evento dati generato dalla porta badge.
void	**KEYB** (Master IPTTS,int isl,int ish , Strframe letsocket,String[] CmaskF) Evento dati generato dalla tastiera o dal video formattato.
void	**DIGIO** (Master IPTTS,int isl,int ish , Strframe letsocket) Evento dati generato dalla porta digital input.
void	**CLK** (Master IPTTS,int isl,int ish , Strframe letsocket) Evento dati generato dall'orologio datario.
void	**ADCONV** (Master IPTTS,int isl,int ish , Strframe letsocket) Evento dati generato dalla porta analog input.
void	**BUTTONS** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF) Evento dati generato dai Bottoni.
void	**RS232C1** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF) Evento dati generato dalla porta seriale RS 232C COM2.
void	**GLOBAL** (Master IPTTS,int isl,int ish , Strframe letsocket) Evento dati generato dall'unita' globale.

void	**GLOBAC** (Master IPTTS,int isl,int ish , Strframe letsocket,String[] CmaskF) Evento risposta generato dall'unita' globale.
void	**manQue** () Evento generato dal thread della classe Ttsman.
void	**RS232D** (Master IPTTS,int isl,int ish ,Strframe letsocket) Evento di diagnostica generato dalla porta seriale RS 232C COM1.
void	**RS232D1** (Master IPTTS,int isl,int ish ,Strframe letsocket) Evento di diagnostica generato dalla porta seriale RS 232C COM2.
void	**SESSIOND** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF) Evento di diagnostica della sessione principale.
void	**SESECO** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF) Evento generato da una sessione ausiliaria in eco.
void	**endConfiguration** (Master IPTTS,int isl) Evento per fine configurazione dello slave IPTTS.TTS[isl].
boolean	**failConfiguration** (Master IPTTS,int isl) Evento per mancata configurazione dello slave IPTTS.TTS[isl];se il valore di ritorno è true lo slave è resettato.
void	**isSlaveOnline** (Master IPTTS,int isl,boolean conn) Evento per cambiamento di stato dello slave IPTTS.TTS[isl];se conn è true lo slave è online.
void	**isMasterOnline** (Master IPTTS,boolean conn) Evento per cambiamento di stato del master IPTTS;se conn è true il master è online.
void	**isMachineOnline** (Machine mach,boolean conn) Evento per cambiamento di stato dell'oggetto mach;se conn è true questo è online.

void	**manExit** () Evento per uscita dal configuratore.
ArrayList	**ExstactDisplay** (String display) Il metodo riceve una transazione dal display formattato,e fornisce un ArrayList che contiene delle coppie costituite dal campo di input e dal campo oggetto associato.
String	**SetDesc** (String display,String subrig,int rig,int col) Il metodo sostituisce su un display formattato la sottostringa subring,alle coordinate di riga (rig) e colonna (col).
String	**SetVal** (String display,ArrayList fieldval) Il metodo su un display formattato,valorizza i campi valore forniti nell'ArrayList fieldval.
String	**SetDate** (int ism,int isl,StrTTS strTTS) Il metodo restituisce la data in formato TTS relativa allo slave IPTTS[ism].TTS[isl].
void	**clrQue** () Il metodo azzera la coda.
Object	**readQue** () Il metodo legge il primo elemento della coda;se la coda e' vuota,restituisce l'oggetto null.
void	**writeQue** (Object elem) Il metodo accoda l'oggetto elem.
<G extends JFrame>	**displayParm** (Point location, int mast,int slave, String typeTTS) Il metodo permette di costruire un proprio form di visualizzazione di dati sul sinottico.
<G extends JFrame>	**displayMach** (Point location, String typeMach) Il metodo permette di costruire un proprio form di visualizzazione di dati sul sinottico.
<G extends JFrame>	**displayUser** () Il metodo permette di costruire un proprio form.

⇐ INDICE

ArrayList	**getAllConf** () Il metodo deve leggere i parametri da un Database.
void	**setAllConf** (ArrayList confTTS) Il metodo deve scrivere i parametri dell'oggetto confTTS in un Database.
ArrayList	**getParmUser** () Il metodo deve leggere i parametri dell'oggetto parmUser da un Database.
void	**setParmUser** (ArrayList parmUser) Il metodo deve scrivere i parametri dell'oggetto parmUser in un Database.
int	**synMasterState** (int mast) Deve restituire lo stato dell'oggetto IPTTS[mast] di classe Master.
int	**synSlaveState** (int mast,int slave) Deve restituire lo stato dell'oggetto IPTTS[mast].TTS[slave] di classe Slave.
int	**synMachineState** (String type) Deve restituire lo stato dell'oggetto Machine identificato da "type".
int	**synConnHost** () Deve restituire lo stato dell'oggetto HostCon.
void	**endApply** () Indica l'attivazione del bottone di Apply del Configuratore_Sinottico, per permettere,ad esempio,l'eventuale disattivazione ed riattivazione di tutti i socket.
void	**RD_parm** (BuildParm parmBuild) Il metodo permette di leggere i parametri da un Database,per un proprio configuratore.
void	**WR_parm** (BuildParm parmBuild) Il metodo permette di scrivere i parametri in un Database,per un proprio configuratore.

⇐ INDICE

Method Detail

initialize

public void **initialize** ()

Il metodo crea un oggetto di classe Ttsman,con riferimento alle variabili di instanza fl_panc,fl_consol e fl_synopt.

---

getTtsManInst

public Ttsman **getTtsManInst** ()

Il metodo restituisce l'oggetto di classe Ttsman creato da initialize().

Returns:

Oggetto Ttsman.

---

setFl_panc

public void **setFl_panc** (boolean fl_panc)

Il metodo assegna la variabile fl_panc.

Parameters:

fl_panc – se "true" abilita il pannello di prova "stato connessine socket", che permette di lanciare il configuratore ed attivare i socket.

---

setFl_consol

public void **setFl_ consol** (boolean fl_consol)

Il metodo assegna la variabile fl_ consol.

Parameters:

fl_consol – se "true" abilita la console.

---

setFl_synopt

public void **setFl_synopt** (boolean fl_synopt)

Il metodo assegna la variabile fl_synopt.

Parameters:

fl_synopt – se "true" abilita il pannello Configuratore_Sinottico, che permette di configurare e visualizzare lo stato dei dispositivi presenti.

⇐ INDICE

getCommState

public CommState **getCommState** ()

Il metodo restituisce l'oggetto di classe CommState creato da initialize().

Returns:

Oggetto CommState.

L'oggetto è utilizzato per le assegnazioni seguenti:

getCommState.setExternMode(true);

getCommState.create();

nell'applicazione Configuratore_Sinottico descritta in precedenza.

---

setVConfigFile

public void **setVConfigFile** (File configFile)

Il metodo definisce il file di configurazione da sovrascrivere al file "ttsparm.txt".

Parameters:

configFile – file di configurazione.

RS232C

public void **RS232C** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa dati e' inviata dalla porta seriale RS 232C COM1.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=0) dell'oggetto rs232c di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

⇐ INDICE

VAC

public void **VAC** (Master IPTTS, String adVAC,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dal canale VAC di indirizzo adVAC.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene il canale.

adVAC – indirizzo dell'elemento di classe VAC dell'array VAC[].

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

VSC

public void **VSC** (Master IPTTS, String adVSC,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dal canale VSC di indirizzo adVSC.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene il canale.

adVSC – indirizzo dell'elemento di classe VSC dell'array VSC[].

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

MACHINE

public void **MACHINE** (Machine mach, Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata da una macchina di tipo Machine.

Parameters:

mach – oggetto di classe Machine relativo.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

⇐ INDICE

parseMachine

public void **parseMachine** (ConnMachine machParm, Object file)

Il metodo si attiva per permettere il parsing o il filtro di un campo.

Parameters:

machParm – oggetto di classe ConnMachine.

file – oggetto da trattare,che se è di classe String riporta una testata(ad es. il nome di un file),altrimenti un record.

Returns:

l'esito dell'operazione.

BARCOD

public void **BARCOD** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dalla porta barcode.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=1) dell'oggetto barcode di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

BADGE

public void **BADGE** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa dati e' inviata dalla porta badge.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=2) dell'oggetto badge di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

⇐ INDICE

KEYB

public void **KEYB** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa dati e' inviata dalla tastiera o dal video formattato.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=4) dell'oggetto keyboard di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

CLK

public void **CLK** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dall'orologio datario.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=7) dell'oggetto orologio di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

⇐ INDICE

DIGIO

public void **DIGIO** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dalla porta digital.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=8) dell'oggetto digital di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

ADCONV

public void **ADCONV** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dalla porta analog.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=9) dell'oggetto adconv di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

BUTTONS

public void **BUTTONS** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa dati e' inviata dai bottoni.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=10) dell'oggetto buttons di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

⇐ INDICE

RS232C1

public void **RS232C1** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa dati e' inviata dalla porta seriale RS 232C COM2.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=11) dell'oggetto rs232c1 di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

GLOBAL

public void **GLOBAL** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa dati e' inviata dall'unità globale.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=15) dell'oggetto global di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

⇐ INDICE

GLOBAC

public void **GLOBAC** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa di risposta e' inviata dall'unità globale o dal master.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico,o = 15 per il master.

ish – indice (=15) dell'oggetto global di classe Hwobject , relativo allo slave fisico, o al master.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

---

manQue

public void **manQue** ()

Il metodo e' attivato dal thread della classe Ttsman.

⇐ INDICE

RS232D

public void **RS232D** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa di diagnostica e' inviata dalla porta seriale RS 232C COM1.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=0) dell'oggetto rs232c di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

RS232D1

public void **RS232D1** (Master IPTTS,int isl,int ish ,Strframe letsocket)

Il metodo si attiva se una stringa di diagnostica e' inviata dalla porta seriale RS 232C COM2.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=11) dell'oggetto rs232c1 di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

GLOBAD

public void **GLOBAD** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa di diagnostica e' inviata dall'unità globale.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

ish – indice (=15) dell'oggetto global di classe Hwobject , relativo allo slave fisico.

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

⇐ INDICE

SESSIOND

public void **SESSIOND** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa di diagnostica e' inviata alla sessione di backup.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice (= 15)

ish – indice (=15)

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

SESECO

public void **SESECO** (Master IPTTS,int isl,int ish ,Strframe letsocket,String[] CmaskF)

Il metodo si attiva se una stringa di eco e' inviata dalla sessione principale o di backup.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice (= 15)

ish – indice (=15)

letsocket – oggetto di classe Strframe relativo al frame ricevuto.

CmaskF – oggetto di classe String[] relativo alle maschere formattate disponibili.

---

endConfiguration

public void **endConfiguration** (Master IPTTS,int isl)

Il metodo avvisa che uno slave fisico è stato configurato e quindi permette di sincronizzare con questo evento eventuali azioni verso di esso.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

⇐ INDICE

failConfiguration

public boolean **failConfiguration** (Master IPTTS,int isl)

Il metodo avvisa che uno slave fisico non è stato configurato e quindi permette di sincronizzare con questo evento eventuali azioni verso di esso;se il valore di ritorno è true lo slave è resettato.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

Returns:

failconf – boolean.

---

isSlaveOnline

public void **isSlaveOnline** (Master IPTTS,int isl,boolean conn)

Il metodo avvisa che uno slave fisico ha perso o ripreso la connessione.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

conn – se true lo slave fisico è online.

---

isMasterOnline

public void **isMasterOnline** (Master IPTTS,boolean conn)

Il metodo avvisa che un master fisico ha perso o ripreso la connessione.

Parameters:

IPTTS – oggetto di classe Master relativo al master fisico a cui appartiene lo slave.

conn – se "true" lo slave fisico è online.

---

isMachineOnline

public void **isMasterOnline** (Machine mach,boolean conn)

Il metodo avvisa che un oggetto mach ha perso o ripreso la connessione.

Parameters:

mach – oggetto di classe Machine.

conn – se "true" lo slave fisico è online.

⇐ INDICE

manExit

public void **manExit** ()

Il metodo avvisa che il configuratore è stato chiuso,e si attiva se setFl_panc(false).

ExstactDisplay

public ArrayList **ExstactDisplay** (String display)

Il metodo riceve una transazione da display formattato,e fornisce un ArrayList che contiene delle coppie costituite dal campo di input e dal campo oggetto associato.

Parameters:

display – contiene la transazione trasmessa dallo slave fisico,composta da campi di input e di oggetto.

Returns:

ArrayList in cui un elemento contiene la stringa del campo di input ed il successivo la stringa del campo oggetto associato .

SetDesc

public String **SetDesc** (String display,String subrig,int rig,int col)

Il metodo sostituisce su un display formattato la sottostringa subring,alle coordinate di riga (rig) e colonna (col).

Parameters:

display – contiene un display formattato;

subrig – contiene la stringa da sostituire;

rig – la coordinata di riga (da 0 a 15);

col – la coordinata di colonna (da 0 a 19);

Returns:

il display formattato modificato.

⇐ INDICE

SetVal

public String **SetVal** (String display, ArrayList fieldval)

Il metodo su un display formattato,valorizza i campi valore forniti nell'ArrayList fieldval.

Parameters:

display – contiene un display formattato;

fielval – contiene i valori di tutti i campi da assegnare.

Returns:

il display formattato modificato.

SetDate

public String **SetDate** (int ism,int isl,StrTTS strTTS)

Il metodo restituisce la data in formato TTS relativa allo slave IPTTS[ism].TTS[isl].)

Parameters:

ism – indice dell'elemento di classe Master dell'array IPTTS[], a cui appartiene lo slave.

isl – indice dell'elemento di classe Slave dell'array TTS[], relativo allo slave fisico.

strTTS – oggetto di classe StrTTS .

Returns:

data e ora nel formato TTS.

---

clrQue

public void **clrQue** ()

Il metodo azzera la coda.

---

readQue

public Object **readQue** ()

Il metodo legge il primo elemento della coda;se la coda e' vuota,restituisce l'oggetto null.

Returns:

Un oggetto.

⇐ INDICE

writeQue

public void **writeQue** (Object elem)

Il metodo accoda l'oggetto elem.

Parameters:

elem – contiene l'oggetto da accodare.

---

displayParm

public <G extends JFrame> G **displayParm** (Object location,int mast,int slave,String typeTTS)

Il metodo permette di costruire un proprio form di visualizzazione di dati sul sinottico.

Parameters:

location – contiene le coordinate del punto di presentazione sul sinottico.

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.
slave – indice dell'array TTS (da 1 a 254) di oggetti di classe Slave.

typeTTS – stringa rappresentante il tipo del dispositivo,i valori possibili sono:

TTS30M;
IF30;
IF302;
IF310;
IF330;
IF340;
IF350;
IF360.

Returns:

G – restituisce un oggetto grafico ereditante la classe JFrame.

---

displayMach

public <G extends JFrame> G **displayMach** (Object location, String typeMach)

Il metodo permette di costruire un proprio form di visualizzazione di dati sul sinottico.

Parameters:

location – contiene le coordinate del punto di presentazione sul sinottico.

typeMach – stringa identificante la macchina.

Returns:

G – restituisce un oggetto grafico ereditante la classe JFrame.

⇐ INDICE

displayUser

public <G extends JFrame> G **displayUser** ()

Il metodo permette di costruire un proprio form.

Returns:

G – restituisce un oggetto grafico ereditante la classe JFrame.

---

getAllConf

public ArrayList **getAllConf** ()

Il metodo permette di leggere i parametri del TTS da un Database,e restituirli in un ArrayList, alla partenza dell'applicazione.Il metodo si attiva tramite setFl_extpar(true) e setFl_synopt(true).

Returns:

ArrayList – contiene tre ArrayList,uno per i parametri di tutti gli oggetti, il secondo per le maschere formattate per un display,ed il terzo per i parametri relativi alla connessione all'Host.

---

setAllConf

public void **setAllConf** (ArrayList confTTS)

Il metodo permette di scrivere i parametri del TTS su un Database, a seguito di una richiesta di salvataggio.Il metodo si attiva tramite setFl_extpar(true) e setFl_synopt(true).

Parameters:

confTTS – contiene tre ArrayList,uno per i parametri di tutti gli oggetti, il secondo per le maschere formattate per un display,ed il terzo per i parametri relativi alla connessione all'Host.

---

getParmUser

public ArrayList <ArrayList < FieldColumn >> **getParmUser** ( )

Il metodo permette di leggere i parametri da un Database,e restituirli in un ArrayList.

Returns:

ArrayList< ArrayList <FieldColumn>> – contiene ArrayList,ognuno composto composto da un massimo di 10 elememti di classe FieldColumn.

⇐ INDICE

setParmUser

public void **setParmUser** (ArrayList <ArrayList < FieldColumn >>parmUser)

Il metodo permette di scrivere righe da 10 parametri generici su un Database.

Parameters:

parmUser – è un ArrayList di ArrayList,i cui elementi sono di classe FieldColumn;

---

synMasterState

public int **synMasterState** (int mast)

Deve restituire lo stato dell'oggetto IPTTS[mast] di classe Master.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

Returns:

Lo stato dell'oggetto relativamente al suo socket:

0= attivo;
1= disattivo;
2= in errore;
3= in recovery;
4= inesistente.

---

synSlaveState

public int **synSlaveState** (int mast,int slave)

Deve restituire lo stato dell'oggetto IPTTS[mast].TTS[slave] di classe slave.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

slave – indice dell'array TTS (da 1 a 254) di oggetti di classe Slave.

Returns:

Lo stato dell'oggetto relativamente al collegamento dello slave fisico relativo:

0= attivo;
1= disattivo;
2= in configurazione;
3= inesistente.

⇐ INDICE

synMachineState

public int **synMachineState** (String type)

Deve restituire lo stato dell'oggetto Machine identificato da "type".

Parameters:

type – stringa identificante la macchina.

Returns:

Lo stato dell'oggetto relativamente al collegamento della macchina:

0= attivo;
1= disattivo.

---

synConnHost

public int **synConnHost** ()

Deve restituire lo stato dell'oggetto HostCon,relativamente al tipo di Host scelto.

Returns:

Lo stato dell'oggetto relativamente al collegamento all'Host:

0= attivo;
1= disattivo;
2= inesistente.

---

endApply

public void **endApply**()

Indica l'attivazione del bottone di Apply del Configuratore_Sinottico,
per permettere,ad esempio,l'eventuale disattivazione ed riattivazione di tutti
i socket.

RD_parm

public void **RD_parm** (BuildParm parmBuild)

Il metodo permette di leggere i parametri del TTS da un Database,e scriverli nelle
strutture interne tramite i metodi della classe BuildParm,alla partenza dell'applicazione.
Il metodo si attiva tramite setFl_extpar(true).

Parameters:

parmbuild – è un oggetto di classe BuildParm.

⇐ INDICE

WR_parm

public void **WR_parm** (BuildParm parmBuild)

Il metodo permette di scrivere i parametri del TTS in un Database,leggendoli dalle strutture interne tramite i metodi della classe BuildParm,a seguito della richiesta di salvataggio tramite la funzioe actSave() della classe Ttsman.
Il metodo si attiva tramite setFl_extpar(true).

Parameters:

parmbuild – è un oggetto di classe BuildParm.

⇐ INDICE

Class Ttsman

---

public class **Ttsman**

La classe **Ttsman** permette di gestire i socket associati ai sistemi TTS attivi, ed il configuratore per la parametrizzazione di tutti i dispositivi TTS.

Vengono costruiti gli oggetti IPTTS di classe master, se configurati.

Rende disponibile un thread,che attiva il metodo manQue della classe StrTTS.

Method Summary	
void	**startConfig** () Lancia il configuratore, se flPanc = false.
void	**setupConfMaster** (int mast, boolean conf, boolean log) Sovrascrive le flag che abilitano l'invio della configurazione agli slave all'avvio dell'applicazione e la creazione del log,relative all'oggetto IPTTS[mast].
void	**actMaster** (int mast, int session) Attiva il socket dell'oggetto IPTTS[mast] configurato, sulla sessione di tipo session.
void	**actAllMasters** () Attiva i socket di tutti gli oggetti IPTTS configurati, sulla sessione principale.
void	**deactMaster** (int mast) Disattiva il socket dell'oggetto IPTTS[mast].
void	**deactAllMasters** () Disattiva i socket di tutti gli oggetti IPTTS configurati.
int	**getStateTtsman** () Restituisce lo stato dell'oggetto di classe Ttsman**.**
int	**getStateMaster** (int mast) Restituisce lo stato dell'oggetto IPTTS[mast] di classe Master.

int	**getStateSlave** (int mast, int slave) Restituisce lo stato dell'oggetto IPTTS[mast].TTS[slave] di classe Slave.
String	**getParmMaster** (int mast, String parm) Restituisce il valore del parametro parm dell'oggetto IPTTS[mast] di classe Master.
String	**getParmSlave** (int mast, int slave,int hwo,String parm) Restituisce il valore del parametro parm dell'oggetto IPTTS[mast].TTS[slave].HWO[hwo] di classe Hwobject.
ArrayList	**getStructULSlave** (int mast, int slave) Restituisce lo stato di configurazione di tutti gli oggetti di classe Hwobject.
String	**getTypeMaster** (int imast) Restituisce il tipo dell'oggetto IPTTS di indice imast.
String	**getTypeSlave** (int imast, int slv) Restituisce il tipo dell'oggetto TTS di indice islv,e referente l'oggetto IPTTS di indice imast.
void	**actSave** () Attiva il metodo WR_parm della classe StrTTS.
void	**startThread** (int time) Attiva il thread con la cadenza definita dalla variabile time,espessa in msec.
void	**stopThread** () Disattiva il thread.
void	**actMachine** (String type) Attiva la connessione verso la macchina identificata da "type".
void	**actAllMachines** () Attiva la connessione verso tutte le macchine configurate.
void	**deactMachine** (String type) Disattiva la connessione verso la macchina identificata da "type".

⇐ INDICE

void	**deactAllMachines** () Disattiva la connessione verso tutte le macchine configurate.
int	**getStateMachine** (String type) Restituisce lo stato della connessione della macchina identificata da “type”.
Machine	**getMachine** (String type) Restituisce un oggetto Machine identificato da “type”.

⇐ INDICE

Method Detail

startConfig

public void **startConfig**()

Lancia il configuratore se flPanc=false

---

setupConfMaster

public void **setupConfMaster** (int mast,boolean conf, boolean log)

Sovrascrive le flag che abilitano l'invio delle configurazioni agli slave all'avvio delle applicazioni e la creazione del log,relative all'oggetto IPTTS[mast].

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master;
conf – se "true" le configurazioni sono inviate;
log – se "true" viene creato un log come file del tipo "date.txt", dove date è la data corrente.

---

actMaster

public void **actMaster** (int mast,int session)

Attiva il socket dell'oggetto IPTTS[mast] di classe Master, con la sessione di tipo session.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.
session – tipo di sessione scelta per il collegamento ad un master TTS:

0= sessione principale;
1= sessione ausiliaria di backup;
2= sessione ausiliaria di eco.
3= sessione ausiliaria di acquisizione slave.

---

actAllmasters

public void **actAllmasters** ()

Attiva i socket di tutti gli oggetti IPTTS configurati, sulla sessione principale.

---

deactMaster

public void **deactMaster** (int mast)

Disattiva il socket dell'oggetto IPTTS[mast] di classe Master.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

⇐ INDICE

deactAllmasters

public void **deactAllmasters** ()

Disattiva i socket di tutti gli oggetti IPTTS configurati.

---

getStateTtsman

public int **getStateTtsman** ()

Restituisce lo stato dell'oggetto di classe Ttsman istanziato.

Returns:

lo stato dell'oggetto relativamente alla lettura dei parametri dal file "parmtts.txt":

0= configurato;
1= non configurato.

---

getStateMaster

public int **getStateMaster** (int mast)

Restituisce lo stato dell'oggetto IPTTS[mast] di classe Master.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

Returns:

Lo stato dell'oggetto relativamente al suo socket:

0= attivo;
1= disattivo;
2= in errore;
3= in recovery;
4= inesistente.

---

getStateSlave

public int **getStateSlave** (int mast,int slave)

Restituisce lo stato dell'oggetto IPTTS[mast].TTS[slave] di classe slave.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

slave – indice dell'array TTS (da 1 a 254) di oggetti di classe Slave.

Returns:

Lo stato dell'oggetto relativamente al collegamento dello slave fisico relativo:

0= attivo;
1= disattivo;
2= in configurazione;
3= inesistente.

⇐ INDICE

getParmMaster

public String **getParmMaster** (int mast,String parm)

Restituisce lo il valore del parametro parm.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

parm – carattere che identifica il parametro da restiuire.

Returns:

parmaster – valore del parametro richiesto.

---

getParmSlave

public String **getParmSlave** (int mast,int slave,int hwo,String parm)

Restituisce il valore del parametro parm.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

slave – indice dell'array TTS (da 1 a 254) di oggetti di classe Slave.

hwo – indice dell'array HWO (da 0 a 15) di oggetti di classe Hwobject.

parm – carattere che identifica il parametro da restiuire.

Returns:

parmslave – valore del parametro richiesto

---

getStructULSlave

public ArrayList **getStructULSlave** (int mast,int slave)

Restituisce lo stato di configurazione di tutti i 16 oggetti di classe Hwobject.

Parameters:

mast – indice dell'array IPTTS (da 0 a 254) di oggetti di classe Master.

slave – indice dell'array TTS (da 1 a 254) di oggetti di classe Slave.

Returns:

ULSlave – oggetto di classe ArrayList, composto da boolean array di quattro elementi:

confUL[0] = se true l'oggetto e' configurato;
confUL[1] = se true l'oggetto e' bloccato a fine configurazione;
confUL[2] = se true l'oggetto riceve un set-up;
confUL[3] = se true l'oggetto non esiste.

⇐ INDICE

getTypeMaster

public String **getTypeMaster** (int imast)

Restituisce il tipo dell'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

Returns:

out – stringa contenente il tipo di master,i valori possibili sono:

TTS30M;
IF30;
IF302;
IF310;
IF330;
IF340;
IF350;
IF360.

---

getTypeSlave

public String **getTypeSlave** (int imast,int islv)

Restituisce il tipo dell'oggetto TTS di indice islv,e referente l'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

Returns:

out – stringa contenente il tipo di slave,i possibili valori sono:

TTS01S;
TTS02S;
TTS10S;
TTS30S;
TTS40S;
TTS50S;
TTS60S.

---

actSave

public void **actSave**()

Attiva il metodo WR_parm della classe StrTTS.

⇐ INDICE

startThread

public void **startThread** (int time)

Attiva il thread con la cadenza definita dalla variabile time.

Parameters:

time – temporizzazione espressa in msec.

---

stopThread

public void **stopThread**()

Disattiva il thread.

---

actMachine

public void **actMachine** (String type)

Attiva la connessione verso la macchina identificata da “type”.

Parameters:

type – codice identificante un oggetto di tipo Machine.

---

actAllMachines

public void **actAllMachines** ()

Attiva la connessione verso tutte le macchine configurate.

---

deactMachine

public void **deactMachine** (String type)

Disattiva la connessione verso la macchina identificata da “type”.

Parameters:

type – codice identificante un oggetto di tipo Machine.

---

deactAllMachines

public void **deactAllMachines** ()

Disattiva la connessione verso tutte le macchine configurate.

⇐ INDICE

getStateMachine

public int **getStateMachine**(String type)

Restituisce lo stato della connessione della macchina identificata da “type”.

Parameters:

type – codice identificante un oggetto di tipo Machine.

Returns:

Lo stato dell’oggetto relativamente alla sua connessione:

0= attivo;
1= disattivo;

---

getMachine

public **Machine** **getMachine** (String type)

Restituisce un oggetto Machine identificato da “type”.

Parameters:

type – codice identificante un oggetto di tipo Machine.

Returns:

oggetto di classe Machine.

⇐ INDICE

Class Strframe

---

public class **Strframe**

La classe incapsula il nome del master (socket) e la stringa ricevuta, essa rende disponibile due variabili di istanza di tipo String:

NAMEm – nome del master;

frame – stringa ricevuta.

⇐ INDICE

Class BuildParm

---

public class **BuildParm**

La classe espone i metodi per leggere o settare i parametri di una struttura TTS.

Method Summary	
void	**setTypeMaster** (int imast,String type) Assegna il tipo all'oggetto IPTTS di indice imast.
void	**setParmMaster** (int imast,String parm,String text) Assegna al parametro parm dell'oggetto IPTTS,di indice imast,il valore text.
void	**setLogMaster**(int imast,String abil) Per l'oggetto IPTTS,di indice imast,se abil = "1" è abilitata la scrittura del file di log.
void	**setConfMaster** (int imast,String abil) Per l'oggetto IPTTS,di indice imast, se abil = "1" è abilitato l'invio dei parametri di configurazione della struttura degli slave ad esso referente,alla partenza dell'applicazione.
void	**setDescMaster** (int imast,String desc) Assegna alla descrizione dell'oggetto IPTTS,di indice imast,la stringa desc.
String	**getTypeMaster** (int imast) Restituisce il tipo dell'oggetto IPTTS di indice imast.
String	**getParmMaster** (int imast,String parm) Restituisce il parm dell'oggetto IPTTS di indice imast.
String	**getLogMaster** (int imast) Restituisce per l'oggetto IPTTS,di indice imast, l'abilitazione alla scrittura del file di log.

⇐ INDICE

String	**getConfMaster** (int imast) Restituisce per l'oggetto IPTTS,di indice imast, l'abilitazione all'invio dei parametri di configurazione della struttura degli slave ad esso referente,all'avvio dell'applicazione.
String	**getDescMaster** (int imast) Restituisce la descrizione dell'oggetto IPTTS di indice imast.
void	**setTypeSlave** (int imast,int islv,String type) Assegna il tipo all'oggetto TTS,di indice islv,referente l'oggetto IPTTS di indice imast.
void	**setExistHwo** (int imast,int islv,int ihwo,String abil) L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è instanziato se abil = "1".
void	**setConfHwo** (int imast,int islv,int ihwo,String abil) L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è configurato se abil = "1".
void	**setLockHwo** (int imast,int islv,int ihwo,String abil) L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è bloccato in input,a fine configurazione,se abil = "1".
void	**setSetupHwo** (int imast,int islv,int ihwo,String abil,String text) L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,riceve la stringa di setup text se abil = "1".
void	**setListHwo** (int imast,int islv,int ihwo,String text) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è definita la stringa text dei parametri da configurare.
boolean	**setParmSlave** (int imast,int islv,int ihwo,String parm,String text,boolean newSlv) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è assegnato il parametro parm con la stringa text e restituisce newSlv = false.

⇐ INDICE

void	**setDescSlave** (int imast,int islv,String desc) Assegna alla descrizione dell'oggetto TTS,di indice islv,e referente l'oggetto IPTTS di indice imast,la stringa desc.
String	**getTypeSlave** (int imast,int islv) Restituisce il tipo dell'oggetto TTS,di indice islv,e referente l'oggetto IPTTS di indice imast.
String	**getExistHwo** (int imast,int islv,int ihwo) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1" se esso esiste.
String	**getConfHwo** (int imast,int islv,int ihwo) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1" se esso è da configurare.
String	**getLockHwo** (int imast,int islv,int ihwo) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1" se esso è da bloccare per l'input a fine configurazione.
String	**getSetupHwo** (int imast,int islv,int ihwo) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1", se esiste un setup,concatenato con "|" + STRINGA DI SETUP.
String	**getListHwo** (int imast,int islv,int ihwo) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituita la stringa elencante i parametri da configurare.
String	**getParmSlave** (int imast,int islv,int ihwo,String parm) Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituita la stringa relativa al parametro parm.
String	**getDescSlave** (int imast,int islv) Restituisce la descrizione dell'oggetto TTS di indice islv,referente l'oggetto IPTTS di indice imast.

⇐ INDICE

void	**setMask** (int index,String type,String mask) Assegna il formato video CmaskF[index] con la stringa mask,per un tipo di slave dotato di display.
String	**getMask** (int index) Restituisce il tipo di slave concatenato con "|" + CmaskF[index].

Method Detail

setTypeMaster

public void **setTypeMaster** (int imast,String type)

Assegna il tipo all'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

type – stringa rappresentante il tipo dell'oggetto,i valori possibili sono:

TTS30M;
IF30;
IF302;
IF310;
IF330;
IF340;
IF350;
IF360.

---

setParmMaster

public void **setParmMaster** (int imast,String parm,String text)

Assegna al parametro parm dell'oggetto IPTTS,di indice imast,il valore text.

Parameters:

imast – indice dell'array IPTTS[];

parm – stringa di un carattere nel range "A,…,"Z";

text – stringa rappresentante il valore del parametro.

⇐ INDICE

setLogMaster

public void **setLogMaster** (int imast,String abil)

Per l'oggetto IPTTS,di indice imast,se abil = "1" è abilitata la scrittura del file di log giornaliero ttslogGGMMAAAA.txt.

Parameters:

imast – indice dell'array IPTTS[];

abil – stringa che assume il valore "0" o "1".

---

setConfMaster

public void **setConfMaster** (int imast,String abil)

Per l'oggetto IPTTS,di indice imast,se abil = "1" è abilitato l'invio dei parametri di configurazione della struttura degli slave ad esso referente,alla partenza dell'applicazione.

Parameters:

imast – indice dell'array IPTTS[];

abil – stringa che assume il valore "0" o "1".

---

setDescMaster

public void **setDescMaster** (int imast,String desc)

Assegna alla descrizione dell'oggetto IPTTS,di indice imast,la stringa desc.

Parameters:

imast – indice dell'array IPTTS[];

desc – stringa che contiene una descrizione.

⇐ INDICE

getTypeMaster

public String **getTypeMaster** (int imast)

Restituisce il tipo dell'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

Returns:

out – stringa contenente il tipo di master.

---

getParmMaster

public String **getParmMaster** (int imast,String parm)

Restituisce il parm dell'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

parm – stringa di un carattere nel range "A,…,"Z";

Returns:

out – stringa contenente il valore del parametro.

---

getLogMaster

public String **getLogMaster** (int imast)

Restituisce per l'oggetto IPTTS,di indice imast, l'abilitazione alla scrittura del file di log.

Parameters:

imast – indice dell'array IPTTS[];

Returns:

out – stringa di un carattere "0" o "1".

---

getConfMaster

public String **getConfMaster** (int imast)

Restituisce per l'oggetto IPTTS,di indice imast, l'abilitazione all'invio dei parametri di configurazione della struttura degli slave ad esso referente,all'avvio dell'applicazione.

Parameters:

imast – indice dell'array IPTTS[];

Returns:

out – stringa di un carattere "0" o "1".

---

getDescMaster

public String **getDescMaster** (int imast)

Restituisce la descrizione dell'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

Returns:

out – stringa contenente la descrizione del master.

---

setTypeSlave

public void **setTypeSlave** (int imast,int islv,String type)

Assegna il tipo all'oggetto TTS,di indice islv,referente l'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

type – stringa rappresentante il tipo dell'oggetto,i valori possibili sono:

TTS01S;
TTS02S;
TTS10S;
TTS30S;
TTS40S;
TTS50S;
TTS60S.

⇐ INDICE

setExistHwo

public void **setExistHwo** (int imast,int islv,int ihwo,String abil)

L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è instanziato se abil = "1".

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

abil – stringa che assume il valore "0" o "1".

---

setConfHwo

public void **setConfHwo** (int imast,int islv,int ihwo,String abil)

L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è configurato se abil = "1".

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

abil – stringa che assume il valore "0" o "1".

---

setLockHwo

public void **setLockHwo** (int imast,int islv,int ihwo,String abil)

L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è bloccato in input,a fine configurazione,se abil = "1".

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

abil – stringa che assume il valore "0" o "1".

⇐ INDICE

setSetupHwo

public void **setSetupHwo** (int imast,int islv,int ihwo,String abil,String text)

L'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,riceve la stringa di setup text se abil = "1".

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

abil – stringa che assume il valore "0" o "1".

text – stringa di setup.

---

setListHwo

public void **setListHwo** (int imast,int islv,int ihwo,String text)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è definita la stringa text dei parametri da configurare.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

text – stringa composta da una sequenza di lettere ("A,…,"Z"),una per ogni parametro.

⇐ INDICE

setParmSlave

public boolean **setParmSlave** (int imast,int islv,int ihwo,String parm,String text,boolean newSlv)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è assegnato il parametro parm con la stringa text e restituisce newSlv = false.

Parameters:

imast – indice dell'array IPTTS[];
islv – indice dell'array TTS[];
ihwo – indice dell'array lunit[];

parm – stringa di un carattere nel range "A,…,"Z";

text – stringa rappresentante il valore del parametro.

newSlv – deve essere 'true' per ogni nuovo Hwobject.

---

setDescSlave

public void **setDescSlave** (int imast,int islv,String desc)

Assegna alla descrizione dell'oggetto TTS,di indice islv,e referente l'oggetto IPTTS di indice imast,la stringa desc.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

desc – stringa che contiene una descrizione.

---

getTypeSlave

public String **getTypeSlave** (int imast,int islv)

Restituisce il tipo dell'oggetto TTS,di indice islv,e referente l'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

Returns:

out – stringa contenente il tipo di slave.

⇐ INDICE

getExistHwo

public String **getExistHwo** (int imast,int islv,int ihwo)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1" se esso esiste.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

Returns:

out – stringa di un carattere "0" o "1".

---

getConfHwo

public String **getConfHwo** (int imast,int islv,int ihwo)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1" se esso è da configurare.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

Returns:

out – stringa di un carattere "0" o "1".

⇐ INDICE

getLockHwo

public String **getLockHwo** (int imast,int islv,int ihwo)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1" se esso è da bloccare per l'input a fine configurazione.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

Returns:

out – stringa di un carattere "0" o "1".

---

getSetupHwo

public String **getSetupHwo** (int imast,int islv,int ihwo)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituito "1", se esiste un setup,concatenato con "|" + STRINGA DI SETUP.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

Returns:

out – stringa di setup.

⇐ INDICE

getListHwo

public String **getListHwo** (int imast,int islv,int ihwo)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituita la stringa elencante i parametri da configurare.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

Returns:

out – stringa composta da una sequenza di lettere ("A,…,"Z"),una per ogni parametro.

---

getParmSlave

public String **getParmSlave** (int imast,int islv,int ihwo,String parm)

Per l'oggetto Hwobject,di indice ihwo,referente gli oggetti IPTTS di indice imast e TTS di indice islv,è restituita la stringa relativa al parametro parm.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

ihwo – indice dell'array lunit[];

parm – stringa di un carattere nel range "A,…,"Z";

Returns:

out – stringa contenente il valore del parametro.

⇐ INDICE

getDescSlave

public String **getDescSlave** (int imast,int islv)

Restituisce la descrizione dell'oggetto TTS di indice islv,referente l'oggetto IPTTS di indice imast.

Parameters:

imast – indice dell'array IPTTS[];

islv – indice dell'array TTS[];

Returns:

out – stringa contenente la descrizione dello slave.

---

setMask

public void **setMask** (int index,String type,String mask)

Assegna il formato video CmaskF[index] con la stringa mask,per un tipo di slave dotato di display.

Parameters:

index – indice dell'array CmaskF[];

type – stringa rappresentante il tipo di slave,i valori possibili sono:

TTS30S;
TTS40S;
TTS50S;
TTS60S;
RFTERM.

---

getMask

public String **getMask** (int index)

Restituisce il tipo di slave concatenato con "|" + CmaskF[index].

Parameters:

index – indice dell'array CmaskF[];

Returns:

out – stringa contenente un formato video ed il tipo di slave relativo.

⇐ INDICE

Class Master

---

public class **Master**

La classe costruisce un client socket, che permette di comunicare con un sistema TTS.

Costruisce un array TTS[] di classe Slave.

Costruisce un array VAC[] di classe Vac.

Costruisce un array VSC[] di classe Vsc.

Method Summary	
Arraylist	**convFromHex** (ArrayList read,String val) Converte stringhe esadecimali in tipi Long, o Integer,o Short,o Byte,o Float,o String,o Boolean.
String	**convToHex** (Object val) Converte in stringa esadecimale, tipi Long, o Integer,o Short,o Byte,o Float,o String,o Boolean.
String	**getType** () Restituisce il tipo dell'oggetto IPTTS.
String	**getDesc** () Restituisce la descrizione dell'oggetto IPTTS.
int	**getInd** () Restituisce l'indice dell'oggetto IPTTS[].
String	**s_cmd** (String text) Trasmette text tramite il socket, come stringa di comando.

⇐ INDICE

Method Detail

convFromHex

public Arraylist **conVar** (ArrayList read,String val)

Converte la stringa val secondo la struttura definita con l'oggetto read.

Parameters:

read – lista dei tipi di variabili (Long, Integer, Short, Byte, Float, String, Boolean);

val – stringa da convertire.

Returns:

out – lista di oggetti, contenente il valore interpretato secondo la struttura definita con l'oggetto read.

---

convToHex

public String **convToHex** (Object val)

Converte in stringa esadecimale l'oggetto val.

Parameters:

val – oggetto da convertire (Long, Integer, Short, Byte, Float, String, Boolean).

Returns:

out – stringa esadecimale convertita.

---

getType

public String **getType** ()

Restituisce il tipo dell'oggetto IPTTS.

Returns:

stringa del tipo di master.

---

getDesc

public String **getDesc** ()

Restituisce la descrizione dell'oggetto IPTTS.

Returns:

stringa della descrizione del master.

⇐ INDICE

getInd

public int **getInd** ()

Restituisce l'indice dell'oggetto IPTTS[].

Returns:

indice del master.

---

s_cmd

public String **s_cmd**(String text).

Trasmette text tramite il socket, come stringa di comando.

Parameters:

text – stringa di testo da trasmettere

Returns:

text – stringa TTS trasmessa.

⇐ INDICE

Class Slave

---

public class **Slave**

La classe costruisce gli oggetti di classe Hwobject:

rs232c;
barcode;
badge;
display;
keyboard;
relè;
buzzer;
orologio;
digital;
adconv;
buttons;
rs232c1
global.

e l'array lunit [] di classe hwobject, con la corrispondenza:

lunit [0]=rs232c;

""

lunit [11]= rs232c1;
lunit [15]= global;

Method Summary	
String	**getType** () Restituisce il tipo dell'oggetto TTS.
String	**getDesc** () Restituisce la descrizione dell'oggetto TTS.

Method Detail

getType

public String **getType** ()

Restituisce il tipo dell'oggetto TTS.

Returns:

stringa del tipo di slave.

---

getDesc

public String **getDesc** ()

Restituisce la descrizione dell'oggetto TTS.

Returns:

stringa della descrizione dello slave.

⇐ INDICE

Class Hwobject

---

public class **Hwobject**

La classe associa un dispositivo TTS al socket del master.

Method Summary	
String	**s_dat** (String text) Trasmette text tramite il socket, come stringa dati.
String	**s_cmd** (String text) Trasmette text tramite il socket, come stringa di comando.
String	**s_cconf** (String text) Trasmette text tramite il socket, come stringa di configurazione; il metodo e' valido solo per modificare i tasti funzione della tastiera.

⇐ INDICE

Method Detail

s_dat

public String **s_dat**(String text).

Trasmette text tramite il socket, come stringa dati.

Parameters:

text – stringa di testo da trasmettere

Returns:

text – stringa TTS trasmessa.

---

s_cmd

public String **s_cmd**(String text).

Trasmette text tramite il socket, come stringa di comando.

Parameters:

text – stringa di testo da trasmettere

Returns:

text – stringa TTS trasmessa.

---

s_cconf

public String **s_cconf**(String text).

Trasmette text tramite il socket, come stringa di configurazione;
il metodo e' valido solo per modificare i tasti funzione della tastiera.

Parameters:

text – stringa di testo da trasmettere

Returns:

text – stringa TTS trasmessa.

⇐ INDICE

Class Vac

---

public class **Vac**

La classe associa un canale VAC al socket del master.

Method Summary	
String	**s_dat** (String adVac, String text) Trasmette text tramite il socket, sul canale VAC di indirizzo adVac.

Method Detail

s_dat

public String **s_dat** (String adVac, String text).

Trasmette text tramite il socket, sul canale VAC di indirizzo adVac.

Parameters:

adVac – indirizzo del canale VAC (stringa di 2 caratteri)

text – stringa di testo da trasmettere

Returns:

text – stringa TTS trasmessa.

⇐ INDICE

Class Vsc

---

public class **Vsc**

La classe associa un canale VSC al socket del master.

Method Summary	
String	**vscRead** (String adVsc, int timer, int block, int offset,**ArrayList** varin) Trasmette text tramite il socket, sul canale VSC di indirizzo adVsc.
String	**vscWrite** (String adVsc, int block, int offset,**ArrayList** varaout) Trasmette text tramite il socket, sul canale VSC di indirizzo adVsc.
Arraylist	**convVar** (String val) Converte la stringa ricevuta dal canale VSC
String	**letGroupIn** (int index) Legge dalla struttura l'elemento index in formato stringa
String	**buildStr** (int len) Costruisce una stringa di lunghezza len.
void	**setVscGrin** (**ArrayList** varin) Assegna l'ArrayList varin.

⇐ INDICE

Method Detail

vscRead

public String **vscRead** (String adVsc, int timer, int block, int offset,**ArrayList** varin)

Il metodo definisce la struttura dei dati da leggere e la cadenza di lettura;
i tipi di variabili sono:

Long	:	integer signed 32 bit
Integer	:	integer signed 32 bit
Short	:	integer signed 16 bit
Byte	:	integer signed 8 bit
Float	:	float signed 32 bit
String	:	string
Boolean	:	boolean array 8 bit

Parameters:

adVsc – indirizzo del canale VSC (stringa di 2 caratteri).

timer – cadenza di lettura a step di 50 msec (da 0 a 255), timer = 0 blocca la lettura.

block – blocco dati (da 0 a 255).

offset – indirizzo di partenza del blocco dati (da 0 a 255).

varin – lista dei tipi di variabili, che deve corrispondere ad un massimo di 128 byte.

Returns:

text – stringa TTS trasmessa.

⇐ INDICE

vscWrite

public String **vscWrite** (String adVsc, int block, int offset,**ArrayList** varaout)

Il metodo definisce la struttura dei dati da scrivere, i tipi di variabili possibili sono:

Long, Integer, Short, Byte, Float, String, Boolean.

Parameters:

adVsc – indirizzo del canale VSC (stringa di 2 caratteri).

block – blocco dati (da 0 a 255).

offset – indirizzo di partenza del blocco dati (da 0 a 255).

varaout – lista dei tipi di variabili, che deve corrispondere ad un massimo di 64byte.

Returns:

text – stringa TTS trasmessa.

---

convVar

public Arraylist **conVar** (String val)

Converte la stringa ricevuta dal canale VSC secondo la struttura definita con il metodo vscRead.

Parameters:

val – stringa ricevuta dal canale VSC

Returns:

groupin – lista di oggetti, contenente il valore letto ed interpretato secondo la struttura definita.

---

letGroupIn

public String **letGroupIn** (int index)

Legge dalla struttura l'elemento indicizzato in formato stringa.

Parameters:

index – indice dell' elemento da convertire in stringa.

Returns:

vartext – stringa contenente il valore dell'elemento.

⇐ INDICE

buildStr

public String **buildStr** (int len)

Fornisce una stringa di lunghezza len.

Parameters:

len – lunghezza della stringa.

Returns:

stringa di lunghezza len.

---

setVscGrin

public void **setVscGrin**(**ArrayList** varin)

Assegna l'ArrayList varin,che definisce la struttura dei dati da leggere .

Parameters:

varin – lista dei tipi di variabili, che deve corrispondere ad un massimo di 128 byte.

⇐ INDICE

Class Machine

---

public class **Machine**

La classe costruisce la connessione che permette di comunicare con la macchina di produzione secondo il protocollo configurato (VAC,VSC,FTP,MS Access,OPC Client).

Method Summary	
boolean	**selectDB** (**ArrayList** <String> field,String filter) Definisce i campi “field” da leggere con la query “filter”.
ArrayList	**structDB** () Restituisce la struttura dei campi definiti.
Object	**castField** (String value,String type) Opera il cast del “value” secondo “type”.
boolean	**putMachine** (String url,**ArrayList** <FieldColumn> outMach) Trasmette i campi alla macchina,secondo la locazione url.

⇐ INDICE

Method Detail

selectDB

public void **selectDB**(**ArrayList** <String> field,String filter)

Definisce i campi “field” da leggere con la query “filter”.

Parameters:

field – lista dei campi da leggere (se null la query si applica a tutti i campi);

filter – query che definisce la modalità della lettura per i vari protocolli:
–per MS Access,si utilizza la classica struttura da Database;
–per FTP si indica il numero di record successivi da accorpare.

Returns:

l’esito della definizione.

---

structDB

public **ArrayList** **structDB**()

Restituisce la struttura dei campi definiti.

Returns:

lista dei campi generalmente di tipo FieldColumn; se OPC Client,è il protocollo configurato,i campi sono tre ArrayList (input,output e Variant Type);

---

castField

public **Object** **castField**(String value,String type)

Restituisce un oggetto di valore “value” e tipo “type”.

Returns:

oggetto di classe “type”.

⇐ INDICE

putMachine

public boolean **putMachine**(String url,**ArrayList** <FieldColumn> outMach)

Trasmette i campi alla macchina,secondo la locazione url.

Parameters:

url	– locazione;
outMach	– lista dei campi da trasmettere; se OPC Client,è il protocollo configurato, il campo row può essere un **Variant Type**,altrimenti deve **essere vuoto**.

Returns:

l'esito della trasmissione.

⇐ INDICE

Class ConnMachine

---

public class **ConnMachine <CN>**

La classe dispone dei campi:

String type	:codice della macchina;
String desc	:descrizione della macchina;
String typeCon	:tipo di protocollo (MS Access, FTP, OPC Client, Portale_RFID, Pallettizzatore_RFID);
CN conn	:ulteriori campi relativi al tipo di protocollo.

⇐ INDICE

Class ObjectDB

---

public class **ObjectDB**

La classe permette di gestire un Database selezionabile tra quelli previsti:MySql, SqlServer,DB2 windows,DB2 i5,Oracle,MS Access,Sybase.

Constructor Summary
ObjectDB (int typeDB,String dataBase, String tabDB,**ArrayList** <FieldColumn> colTab, String ipServer, **Precision** precision)
ObjectDB (int typeDB,String dataBase, String tabDB,**ArrayList** <FieldColumn> colTab, String ipServer, String user, String password,**Precision** precision)

Method Summary	
boolean	**conDB** () Apre la connessione verso il Database impostato.
boolean	**cloDB** () Chiude la connessione verso il Database impostato.
boolean	**dropTableDB** () Cancella la tabella impostata.
boolean	**insertDB** (**ArrayList** <FieldColumn> field,boolean notEqual) Inserisce una riga nella tabella impostata.
boolean	**updateDB** (**FieldColumn** field, **FieldColumn** key) Aggiorna il campo “field” secondo il campo “key” nella tabella impostata.
ArrayList <ArrayList>	**letTable** (**ArrayList** <String> fields,String filter) Legge dalla tabella impostata i campi “fields”,con la query “filter”.
boolean	**deleteRowDB** (String field, **Object** value) Cancella la riga che contiene il campo di nome “field” e valore “value” nella tabella impostata.

⇐ INDICE

Constructor Detail

ObjectDB

Definisce per un Database le credenziali di connessione ed una tabella,mediante le variabili di instanza:

- **typeDB**,seleziona il tipo di Database,esso può assume i valori:

```
typeMySql = 0;
typeSqlServer = 1;
typeDB2 = 2;
typeDB2i5 = 3;
typeOracle = 4;
typeMsAccess = 5;
typeSybase = 6;
```

- **dataBase**,è il nome del Database definito sul server(“” per DB2 i5 e MS Access);

- **tabDB**,è il nome della tabella che è gestita dai metodi;per DB2 è del tipo “schema.nome”;per DB2 i5 “schema” è la “libreria”;

- **colTab**,è la lista dei campi di una riga;il primo campo deve essere di tipo integer ed è trattato come “primary key” autoincrementante a partire da 1;

- **ipServer**,è l’IP del server o l’URL;

- **user**,è l’utente;

- **password**,è la password;

- **precision**,è un ogetto di classe Precision che definisce i parametri per la risoluzione dei tipi previsti.

Valgono gli esempi seguenti:

- MySql

```
//Server remoto
objectDB = new ObjectDB(ObjectDB.typeMySql,"tts","master",columnTab,
                        "192.168.0.131","aniello","nello",precision);
//Server locale
objectDB = new ObjectDB(ObjectDB.typeMySql,"tts","master",columnTab,
                        "localhost","aniello","nello",precision);
```

- SqlServer

```
//Server remoto
objectDB = new ObjectDB(ObjectDB.typeSqlServer,"PHS","collect",columnTab,
                        "192.168.100.3","PHS","PH2013omba",precision);

//Server Express locale
objectDB = new ObjectDB(ObjectDB.typeSqlServer,"PHS","collect",columnTab,
                        "localhost:1433;integratedSecurity=true”,
                        precision);
```

```
//Server Express remoto
objectDB = new ObjectDB(ObjectDB.typeSqlServer,"PHS","collect", columnTab,
                        "AIOVINE\\SQLEXPRESS;integratedSecurity=true”,
                        precision);
```

- DB2

```
//Server remoto
objectDB = new ObjectDB(ObjectDB.typeDB2,"PHS","tts.collect",columnTab,
                        "192.168.0.131","db2admin","nello”, precision);

//Server locale
objectDB = new ObjectDB(ObjectDB.typeDB2,"PHS","tts.collect",columnTab,
                        "localhost","db2admin","nello”,precision);
```

- DB2 i5

```
objectDB = new ObjectDB(ObjectDB.typeDB2i5,"","tts.collect",columnTab,
                        "192.168.0.102","nello","nello”,precision);
```

- Oracle

```
objectDB = new ObjectDB(ObjectDB.typeOracle,"XE","collect",columnTab,
                        "192.168.0.131","system","nello”,precision);
```

- MS Access

```
//Server MS Access locale
String dbAccess = "C:/Documents and Settings/Aniello " +
                  "Iovine/Documenti/PHS.mdb";
objectDB = new ObjectDB(ObjectDB. typeMsAccess,"","collect",columnTab,
                        dbAccess,precision);

//Server MS Access remoto su PC
String dbAccess = "//192.168.0.131/Documenti/PHS.mdb";

//Server MS Access remoto
String dbAccess = "//192.168.101.199/d/Minosse/DATA/CAM.mdb";

objectDB = new ObjectDB(ObjectDB. typeMsAccess,"","PROD_TIME",columnTab,
                        dbAccess,"CNC1090471//omba","omba2013",precision);
```

- Sybase

```
objectDB = new ObjectDB(ObjectDB.typeSybase,"PHS","collect",columnTab,
                        "192.168.0.131","nello","nello”,precision);
```

⇐ INDICE

Method Detail

conDB

public boolean **conDB**()

Apre la connessione verso il Database impostato.

Returns:

l'esito dell'apertura della connessione.

---

cloDB

public boolean **cloDB**()

Chiude la connessione verso il Database impostato.

Returns:

l'esito della chiusura della connessione.

---

dropTableDB

public boolean **dropTableDB**()

Cancella la tabella impostata.

Returns:

l'esito della cancellazione della tabella impostata.

---

insertDB

public boolean **insertDB**(**ArrayList** <FieldColumn> field,boolean notEqual)

Inserisce una riga nella tabella impostata.

Parameters:

field – lista dei campi da inserire;

notEqual – se "true" la riga è inserita se non sono presenti i campi della lista.

Returns:

l'esito dell'inserimento della riga nella tabella impostata.

⇐ INDICE

updateDB

public boolean **updateDB** (**FieldColumn** field, **FieldColumn** key)

Aggiorna il campo “field” secondo il campo “key” nella tabella impostata.

Parameters:

field – campo da aggiornare;

field – campo per la query;

Returns:

l’esito dell’update del campo nella tabella impostata.

---

letTable

public **ArrayList** <ArrayList> **letTable** (**ArrayList** <String> fields,String filter)

Legge dalla tabella impostata i campi “fields”,con la query “filter”.

Parameters:

fields – campi da leggere;

filter – stringa che definisce la query;

Returns:

lista delle righe lette nella tabella impostata.

---

deleteRowDB

public boolean **deleteRowDB**(String field, **Object** value)

Cancella la riga che contiene il campo di nome “field” e valore “value” nella tabella impostata.

Parameters:

field – il nome del campo;

value – valore del campo.

Returns:

l’esito dellla cancellazione della riga nella tabella impostata.

⇐ INDICE

Class FieldColumn

---

public class **FieldColumn**

La classe dispone dei campi:

String row	:riga di appartenenza;
String nameField	:nome del campo;
Object objField	:valore del campo.

⇐ INDICE

Class Precision

---

public class **Precision**

La classe dispone dei campi:

int size	:dimensione massima del tipo stringa(varchar);default 40;
int precision	:numero massimo di interi del tipo decimal(numeric);default 30;
int decimal	:numero massimo di decimali;default 10.

⇐ INDICE

Class ExternDB

---

public class **ExternDB**

La classe implementa i metodi setAllConf e getAllConf,costruendo le tabelle di configurazione sul Database scelto;è obbligatorio instanziare l'oggetto externDB della classe StrTTS:

externDB = new ExternDB(…….);

l'instanza deve precedere il metodo initialize().

Constructor Summary
ExternDB (int typeDB,String dataBase, String scheme, String ipServer, String user, String password)

⇐ INDICE

Class StructIniDB

---

public class **StructIniDB**

La classe dispone dei campi:

int typeDB : seleziona il tipo di Database;
String database : è il nome del Database definito sul server;
String scheme : è lo schema per i Database che lo richiedono;
String ipserver : è l'IP del server o l'URL;
String user : è l'utente;
String password : è la password.

Essa è instanziata nella classe StrTTS che rende disponibile l'oggetto structIniDB, utilizzabile per una connessione DB.

Vale lo schema seguente:

DB_Type	Database	Schema_Libreria	Tabella	IP_URL	User	Password
MySql	x		x	x	x	x
SqlServer	x		x	x	x	x
DB2_Win	x	x	x	x	x	x
DB2_I5		x	x	x	x	x
Oracle	x		x	x	x	x
MsAccess			x	x	x	x
Sybase	x		x	x	x	x

⇐ INDICE

Class HostVAC

---

public class **HostVAC**

La classe permette di gestire il collegamento tramite un canale VAC di un master TTS30M.

Constructor Summary
HostVAC (String ipMaster, String portVAC)

Method Summary	
void	**txVAC** (String text) trasmette la stringa text sul canale VAC.
String	**getVAC** () riceve una stringa dal canale VAC.

Constructor Detail

HostVAC

Crea un socket UDP mediante le variabili di instanza:

- **ipMaster**,IP del master TTS30M;
- **portVAC**,porta UDP configurata sul master.

Le stringhe ricevute sono disponibili in una coda.

Method Detail

txVAC

public void **txVAC**(String text)

Trasmette la stringa text sul canale VAC.

Parameters:

text – stringa da trasmettere.

⇐ INDICE

getVAC

public String **getVAC**()

Restituisce una stringa estraendola dalla coda.

Returns:

Una stringa di lunghezza > 0 se la coda non è vuota.

⇐ INDICE

Class StructIniVAC

---

public class **StructIniVAC**

La classe dispone dei campi:

String ipMaster : è l'IP del master TTS30M;
String portVAC : è la porta UDP configurata sul master.

Essa è instanziata nella classe StrTTS che rende disponibile l'oggetto structIniVAC, utilizzabile per una connessione VAC.

⇐ INDICE

Class SocketTCP

---

public class **SocketTCP <EV>**

La classe permette di gestire una connessione TCP/IP in modalità client.

Constructor Summary
SocketTCP ()

Method Summary	
boolean	**inizialize** (String ipServer,Integer portServer,String name, **EV** event) inizializza l'oggetto ed attiva la connessione verso il server identificato dalle variabili ipServer e portServer.
void	**txSocket** (String text) trasmette la stringa text sulla connessione.
String	**getSocket** () riceve una stringa dalla connessione.
Socket	**letSocket ()** restituisce un oggetto di classe Socket.
int	**getState ()** restituisce lo stato della connessione .
void	**resCon()** chiude e fa ripartire la connessione.
void	**killer ()** distrugge l'oggetto.

Constructor Detail

SocketTCP

Crea un socket TCP.
Le stringhe ricevute sono disponibili in una coda.

⇐ INDICE

Method Detail

inizialize

public boolean **inizialize**(String ipServer,Integer portServer,String name, **EV event**)

inizializza l'oggetto ed attiva la connessione verso il server.

Parameters:

ipServer – IP del server;

portServer – porta TCP del server;

name – nome associato all'oggetto;

event – oggetto possessore del metodo di callback consock.

Returns:

Esito dell'apertura della connessione.

---

txSocket

public void **txSocket**(String text)

Trasmette la stringa text sulla connessione.

Parameters:

text – stringa da trasmettere.

---

getSocket

public String **getSocket**()

Restituisce una stringa estraendola dalla coda.

Returns:

Una stringa di lunghezza > 0 se la coda non è vuota.

⇐ INDICE

letSocket

public **Socket** **letSocket**()

Restituisce un oggetto di classe Socket.

Returns:

Oggetto di classe Socket.

---

consock

public void **getSocket**(String name,Boolean conn)

Il metodo viene chiamato al variare dello stato della connessione;esso è reso disponibile dall'oggetto **event**.

Parameters:

name – nome associato all'oggetto;

conn – connessione attiva o disattiva.

---

getState

public int **getState**()

Restituisce lo stato della connessione.

Returns:

0 connessione attiva;
1 connessione disattiva;
2 connessione inesistente.

---

resCon

public void **recCon**()

chiude e fa ripartire la connessione.

---

killer

public void **killer**()

distrugge l'oggetto.

⇐ INDICE

Class ServerTCP

---

public class **ServerTCP <EV>**

La classe permette di gestire una connessione TCP/IP in modalità server.

Constructor Summary
ServerTCP ()

Method Summary	
boolean	**inizialize** (Integer portServer,String name, **EV** event) inizializza l'oggetto ed attiva la connessione in modalità "listening" sulla porta portServer.
void	**txSocket** (String text) trasmette la stringa text sulla connessione.
String	**getSocket** () riceve una stringa dalla connessione.
Socket	**letSocket ()** restituisce un oggetto di classe Socket.
int	**getState ()** restituisce lo stato della connessione .
void	**resCon()** chiude e fa ripartire la connessione.
void	**killer ()** distrugge l'oggetto.

Constructor Detail

ServerTCP

Crea un server TCP.
Le stringhe ricevute sono disponibili in una coda.

⇐ INDICE

Method Detail

inizialize

public boolean **inizialize**(Integer portServer,String name, **EV event**)

inizializza l'oggetto ed attiva la connessione verso il server.

Parameters:

portServer – porta TCP del server;

name – nome associato all'oggetto;

event – oggetto possessore del metodo di callback consock.

Returns:

Esito dell'apertura della connessione.

---

txSocket

public void **txSocket**(String text)

Trasmette la stringa text sulla connessione.

Parameters:

text – stringa da trasmettere.

---

getSocket

public String **getSocket**()

Restituisce una stringa estraendola dalla coda.

Returns:

Una stringa di lunghezza > 0 se la coda non è vuota.

⇐ INDICE

letSocket

public **Socket** **letSocket**()

Restituisce un oggetto di classe Socket.

Returns:

Oggetto di classe Socket.

---

consock

public void **getSocket**(String name,Boolean conn)

Il metodo viene chiamato al variare dello stato della connessione;esso è reso disponibile dall'oggetto **event**.

Parameters:

name – nome associato all'oggetto;

conn – connessione attiva o disattiva.

---

getState

public int **getState**()

Restituisce lo stato della connessione.

Returns:

0 connessione attiva;
1 connessione disattiva;
2 connessione inesistente.

---

resCon

public void **recCon**()

chiude e fa ripartire la connessione.

---

killer

public void **killer**()

distrugge l'oggetto.

⇐ INDICE

Class SocketUDP

public class **SocketUDP**

La classe permette di gestire una connessione UDP/IP.

Constructor Summary
SocketUDP (String ipUDP,portUDP)

Method Summary	
void	**txUDP** (String text) trasmette la stringa text sulla connessione.
String	**geUDP** () riceve una stringa dalla connessione.
int	**getState ()** restituisce lo stato della connessione .
void	**killer ()** distrugge l'oggetto.

⇐ INDICE

Constructor Detail

SocketUDP

Crea un socket UDP mediante le variabili di instanza:

- **ipUDP**,IP del socket;
- **portUDP**,porta UDP del socket.

Le stringhe ricevute sono disponibili in una coda.

Method Detail

txUDP

public void **txUDP**(String text)

Trasmette la stringa text sulla connessione.

Parameters:

text – stringa da trasmettere.

---

getUDP

public String **getUDP**()

Restituisce una stringa estraendola dalla coda.

Returns:

Una stringa di lunghezza > 0 se la coda non è vuota.

---

getState

public int **getState**()

Restituisce lo stato della connessione.

Returns:

0 connessione attiva;
1 connessione disattiva;
2 connessione inesistente.

⇐ INDICE

killer

public void **killer**()

distrugge l’oggetto.

⇐ INDICE

Class SocketHTTPS

---

public class **SocketHTTPS**

La classe permette di gestire una connessione TCP/IP certificata TLS su cui appoggiare un layer HTTP autenticato.

Constructor Summary
SocketHTTPS (**Properties** properties)

Method Summary	
Socket	**getSocketSSL** () apre la connessione verso il Web server.
void	**closeSocketSSL** () chiude la connessione.
void	**setSocketSSL** (**Properties** properties) assegna le properties.

Constructor Detail

SocketHTTPS

prepara un socket SSL mediante le poperties,in particolare gli store dei certificati:

- **host**: URL del Web server;
- **hostport**: porta TCP del Web server;
- **pathTrutStore**: percorso del TrustStore;
- **trustPass**: password del TrustStore;
- **pathKeyStore**: percorso del KeyStore;
- **keyPass**: password del KeyStore.

⇐ INDICE

Method Detail

getSocketSSL

public Socket **getSocketSSL**()

Attiva e restituisce un socket SSL.

Returns:

Il socket SSL.

---

closeSocketSSL

public void **closeSocketSSL**()

Chiude la connessione.

---

setSocketSSL

public void **setSocketSSL**(**Properties** properties)

Assegna le properties.

Parameters:

Properties – properties della connessione.

⇐ INDICE

Class SocketUPnP

---

public class **SocketUPnP <EV>**

La classe permette il discovery di un dispositivo di tipo MediaRender (protocollo DLNA), identificando “services” e “actions”.

Constructor Summary
SocketUpnp ()

Method Summary	
boolean	**inizialize** (String ipClient,String name, **EV** event) inizializza l’oggetto ed avvia il discovery del dispositivo con IP ipClient.

Constructor Detail

SocketUPnP

Crea gli oggetti necessari per il Discovery e rende disponibili i metodi per ottenere tutti gli oggetti disponibili.

Method Detail

inizialize

public boolean **inizialize**(String ipClient,String name, **EV event**)

inizializza l’oggetto ed avvia il discovery del dispositivo con IP ipClient.

Parameters:

ipClient – IP del client MediaRender;

name – nome associato all’oggetto;

event – oggetto possessore del metodo di callback consock.

Returns:

Esito dell’apertura della connessione.

⇐ INDICE

consock

public void **getSocket**(String name,Boolean conn)

Il metodo viene chiamato se il discovery ha esito positivo;esso è reso disponibile dall'oggetto **event**.

Parameters:

name – nome associato all'oggetto;

conn – connessione attiva o disattiva.
